

A Framework for the Requirements Analysis of Safety-Critical Computing Systems

Amer Saeed

Ph.D Thesis

**The University of Newcastle upon Tyne
Computing Laboratory**

September 1990

NEWCASTLE UNIVERSITY LIBRARY

089 61630 X

Thesis L3785

Digital computers are increasingly being used in safety-critical applications (e.g., avionics, chemical plant and railway systems). The main motivations for introducing computers into such environments are to increase performance, flexibility and efficiency. However, the cost to safety in achieving these benefits using computing systems is unclear. The general class of systems considered in this thesis are *process control systems*. More specifically the thesis examines the class of *safety-critical computing systems* which are a component of a process control system that could cause or allow the overall system to enter into a hazardous state.

This thesis investigates the role of *formal methods* in safety-critical computing systems. The phase of system development considered is *requirements analysis*. Experience in safety-critical systems has shown that errors in the identified requirements are one of the major causes of mishap. It is argued that to gain a complete understanding of such computing systems, the requirements of the overall system and the properties of the environment must be analyzed in a common formal framework. A *system development model* based on the *separation* of safety and mission issues is discussed, which highlights the *essential specifications* that must be produced during requirements analysis. A formal model for the representation of these essential specifications is presented. The semantics of this formal model are based on the notion of a *system history*. To structure the specifications expressed by this formal model the concept of a *mode* is introduced.

This thesis suggests that for a formal model to be useful during requirements analysis a related *systematic methodology*, which provides comprehensive guidelines for the analysts who use the model must be made available. An appropriate methodology, based upon the system development model, which incorporates some traditional system safety techniques is described. Overall, the thesis presents a *framework* for requirements analysis by providing a system development model, formal model and related development methodology. An example of how this framework can support requirements analysis is presented in the appendices B and C.

Acknowledgements

Firstly, my thanks go to my supervisor, Professor Tom Anderson, who introduced me to the area of safety-critical systems, and in addition made many helpful comments upon the content of this thesis. I would also like to thank Professor Brian Randell and Dr Maciej Koutny for their help in the early years of my research. In particular I would like to thank Dr Maciej Koutny for his suggestions on the formal notation.

Thanks must also go to my colleague Mr R de Lemos for our discussions on safety-critical systems. Many other members of the Computing Laboratory too numerous to mention individually have also made my tenure here more pleasant. Thanks to you all.

Finally the support and patience of my parents throughout the years of this research deserves a special mention.

Financial support for the work described in this thesis was provided by a grant from the UK Science and Engineering Research Council and an Alvey grant in software reliability (Alvey Software Reliability Project SE/072).

Table of Contents

1

Introduction 1

1.1. Background 1

1.2. Safety 3

1.2.1. System Safety 5

1.2.2. Regulation and Legislation 6

1.3. Process Control Systems 7

1.3.1. Operator 8

1.3.2. Physical Process 8

1.3.3. Controller 9

1.3.4. Controller Components 9

1.4. Safety in Computer Based Systems 11

1.4.1. Reliability 12

1.4.2. Safety 12

1.4.3. Safety-critical Computing Systems 13

1.5. Obstacles to Computers in Safety-Critical Systems 13

1.5.1. Intrinsic Obstacles 14

1.5.2. Application Obstacles 17

1.6. System Lifecycle 19

1.7. Aims 22

1.8. Overview 23

2

Requirements Analysis 25

2.1. Introduction 25

2.2. Role of Requirements Specification 26

2.2.1. Requirements Specification Viewpoints 28

2.3. Structured Requirements Analysis 30

2.3.1. Development Model 30

2.3.1.1. Separation of Issues 30

2.3.1.2. Safety-Critical System Structure 32

2.3.1.3. Requirements Phases 37

2.3.2. Formal Model 40

2.3.2.1. Advantages of Formal Models 40

2.3.3. Development Programme 46

2.3.3.1. Development Methodology	47
2.3.3.2. Requirements Analysis Team	48
2.4. Working Example	51
2.5. Summary	53
3	
<i>Basic Concepts</i>	55
3.1. Time	55
3.1.1. Time Points	58
3.1.2. Time Intervals	58
3.1.3. System Lifetime	61
3.2. State Variables	61
3.2.1. Variable Ranges	63
3.3. Variable Categories	65
3.4. System History	68
3.4.1. History Function	69
3.5. History Descriptions	70
3.5.1. Variable Class Relations	71
3.5.2. Invariant Relations	74
3.5.3. History Relations	77
3.5.4. Comparison of Relations	80
3.5.5. History Description Sets	81
3.6. Clocks	82
3.7. Real-time Satisfaction Conditions	83
3.7.1. System Predicates	83
3.7.2. Point Satisfaction	84
3.7.3. Interval Satisfaction	85
3.7.4. Events	87
3.7.5. Time Bound Constraints	90
3.7.6. Termination Predicate	91
3.8. Summary	92
4	
<i>Mode Theory</i>	94
4.1. Modes	94
4.1.1. History Graphs	97
4.1.2. Mode Properties	98
4.1.3. Mode Relationships	100

4.1.4. Mode Categories	101
4.1.5. Mode Consistency	103
4.1.6. Mode Limitations	106
4.1.7. Mode Benefits	107
4.2. Mode Sequences	108
4.2.1. Mode Sequence Properties	109
4.2.2. Mode Sequence Relationships	111
4.2.3. Mode Sequence Consistency	111
4.2.4. Mode Sequence Example	114
4.2.5. Mode Sequence Set	115
4.2.6. Mode Sequence Limitations	116
4.3. Mode Graphs	118
4.3.1. Mode Graph Properties	121
4.3.2. Mode Graph Components	121
4.3.3. Complete Mode Graphs	123
4.3.4. Consistent Mode Graphs	124
4.3.5. Mode Graph Relationships	126
4.3.6. Mode Graph Categories	127
4.3.7. Predicate Mode Graph	130
4.4. Summary	131
 5	
<i>Real World Specifications</i>	<i>133</i>
5.1. Disaster Set	133
5.2. Safety Real World Description	134
5.3. Hazard Specification	135
5.4. Safety Real World Specification	138
5.5. Mission Real World Description	139
5.6. Mission Real World Specification	140
5.6.1. Mission Phase Specification	141
5.7. Summary	144
 6	
<i>Controller Specifications</i>	<i>146</i>
6.1. Safety Environment Description	146
6.2. Safety Controller Specification	148
6.2.1. Safety Controller Behaviour Structure	148
6.2.2. Start Up Phase	150

6.2.3. Monitor Phase	151
6.2.4. Recovery Phase	152
6.2.5. Reset Phase	155
6.2.6. Shut Down Phase	156
6.2.7. End Phase	156
6.3. Mission Environment Description	158
6.3.1. Relation Classes	158
6.3.2. Monitor Relations	160
6.4. Mission Controller Specification	161
6.4.1. Mission Controller Behaviour Structure	162
6.5. Summary and Conclusions	169
7	
<i>Real World Analysis</i>	171
7.1. Introduction	171
7.2 Initial Real World Description Analysis	174
7.2.1. Production Guidelines	174
7.3. Disaster Analysis	180
7.3.1 Disaster Identification	180
7.3.2. Validation Guidelines	181
7.4. Hazard Specification Analysis	182
7.4.1 Hazard Identification	182
7.4.2. Validation Guidelines	184
7.4.3 Hazard Elimination	185
7.4.4. Complete Hazard Assumption	185
7.5. Safety Real World Description Analysis	186
7.5.1. Construction Guidelines	186
7.5.2. Validation Guidelines	188
7.6. Safety Real World Specification Analysis	189
7.7. Mission Real World Specification Analysis	190
7.7.1. Mission Phase Specification Analysis	190
7.7.2. Mission Real World Specification Analysis	195
7.8. Mission Real World Description Analysis	199
7.8.1. Construction Guidelines	199
7.8.2. Mission Real World Specification Checks	201
7.8.3. Mission Validation	202
7.9. Combination of Analysis	203
7.10. Summary	203

8

Controller Analysis 206

8.1. Introduction 206

8.2. Safety Environment Description Analysis 208

8.3. Safety Controller Specification Analysis 210

8.3.1. Safety Verification. 211

8.3.2 Safety Controller Specification Development Methodology 215

8.3.3. Safety Controller Specification Methodology Theorems 226

8.4. Mission Environment Description Analysis 229

8.5. Mission Controller Specification Analysis 231

8.5.1 Mission Verification. 234

8.5.2. Mission Controller Development Strategy 245

8.5.3. Mission Controller Specification Theorems 249

8.6. Summary and Conclusions 251

9

Summary and Conclusions 254

9.1. Thesis Summary 254

9.2. Evaluation and Conclusions 261

9.3. Future Work 263

9.3.1. Extensions to the Framework 264

9.3.2. Investigation of Other Formalisms 266

References

Appendix A

Appendix B

Appendix C

List of Figures

Figure 1.1. Process Control System Components	8
Figure 1.2. Controller Components	9
Figure 1.3. Controller Interfaces	11
Figure 1.4. Simplified System Life Cycle	19
Figure 2.1. Role of Requirements Specification	28
Figure 2.2. Safety–Critical System Structure	32
Figure 2.3. Safety–Critical Controller Components	34
Figure. 2.4. Safety–Critical Controller Interfaces	35
Figure 2.5. Requirement Analysis Phases	37
Figure 2.6. Reaction Vessel	53
Figure 3.1. An Evolution	68
Figure 3.2. A Partial Ordering of Description Relations and Ranges Relations	81
Figure 4.1. History Graphs	98
Figure 4.2. Graphical Argument for Lemma 4.1.	99
Figure 4.3. Mode Graphs	120
Figure 4.4. Mode Graph Isomorphism	127
Figure 5.1. Reaction Vessel Mission Real World Specification Structure	142
Figure 6.1. Reaction Vessel Safety Controller	147
Figure 6.2. General Safety Controller Specification Structure	149
Figure 6.3. Reaction Vessel Safety Controller Specification Structure	157
Figure 6.5. Reaction Vessel Mission Controller	161
Figure 6.4. Reaction Vessel Mission Controller Specification Structure	163
Figure 7.1. Real World Analysis	173
Figure 7.2. Real World Specification Analysis	173
Figure 7.3 Initial Real World Description Analysis	179
Figure 7.4 Safety Real World Description Construction	188
Figure 7.5. High–Level Phase Graph	192
Figure 7.6. Mission Real World Specification Construction	199
Figure 7.7. Mission Real World Description Construction	201
Figure 8.1. Controller Analysis	206
Figure 8.2. Controller Specification Analysis	207
Figure 8.3. General Structure of Safety Controller	210

Figure 8.4. Picture of Lemma 8.1. 233

Figure 8.5. Postulated Check for Start of Sequence rs 235

Figure 8.6. Postulated Check for a Typical Mode in Sequence rs 235

Figure 8.7. Postulated Check for End of Sequence rs 236

Figure 8.8. History Graph of $PS(i)$ 239

Figure 8.9. Time bounds and Delay Function 241

List of Tables

Table 2.1: Organisational Roles in Requirements Analysis 51

Table 3.1: State Variables of Reaction Vessel 62

Table 3.2: Variable Ranges of Reaction Vessel 64

Table 3.3: Class and Categories of Reaction Vessel Variables 74

Table 3.4: Reaction Vessel Description Relations 80

Table 5.1: Relations of Safety Real World Description 135

Table 5.2: Relationships of Mission Real World Description 140

Table 6.1: Controller Relationships of Safety Environment Description 147

Table 6.2: Controller Relationships of Mission Environment Description 160

Table 7.1: Disaster Analysis of Reaction Vessel 181

Chapter 1 - Introduction

1.1. Background

Digital computers are increasingly being used in safety-critical applications (e.g., air traffic control [Cris90], avionics [Rouq86] and railway systems [Theu86a]). A natural reluctance to introduce complex and uncertain factors into these systems had previously kept computers out of most safety-critical loops. However, with the massive increase in computer processing power per dollar [Gray88] over the last decade, many organisations now feel that the potential advantages of using computers often outweighs any nervousness. The functions which computers perform in safety-critical systems are now changing. Until recently when computers were found in such systems, they were only given control over non-critical functions [Leve86]. However, computers are now being given direct control over critical functions, in some cases without any effective back-up facilities [Leve86]. Thus, in these cases, total reliance is placed on the computers.

The increased use of computers in safety-critical systems has led to a growing public awareness about the dangers of introducing computers into safety-critical systems. This is illustrated by the topic being raised in the general media [Equ90, Morg89].

In the late 1970s the main motivations for introducing computers into safety-critical environments were to increase performance, flexibility and efficiency [Long77, Uram77]. More recently, it has been argued that computers can also increase safety [Rous81], however at present there is no scientific evidence to support this supposition. The main reason for the lack of suitable evidence arises from the fact that safety-critical systems tend to have reliability requirements in the region of 10^{-10} failures/hour, e.g., NASA stipulates the requirement of “at most 10^{-9} chance of failure over a 10 hour flight” [Dunh81]. These reliability requirements are far higher than can be currently demonstrated for complex computing systems, the best that can be claimed for systems using current methods is of the order 10^{-5} failures/hour [Mose90].

Unlike processing power, there has not been a massive increase in the quality of software produced over the last decade. This has resulted in a climate where the reliability

of software developed using the best current methods is still orders of magnitude below the levels required for safety-critical systems [Dunh81, Tayl89].

Safety-critical systems are kept under strict safety standards by government licensing bodies, such as the Health and Safety Executive [Roys77, ACAR86]. This introduces a further obstacle to the introduction of computers into safety-critical systems, for there are currently no practical techniques which can measure high-levels of reliability for software, of the order 10^{-9} failures/hour. The best current techniques can only be used to certify (i.e., predict) software reliability in the region of 10^{-5} to 10^{-6} failures/hour. In fact, the task of accurately predicting software reliabilities may be more difficult than constructing highly reliable systems [Tayl89, Litt85]. Most efforts to quantify software reliability usually relate to determining the number of faults (bugs) which exist in a program. However, the reliability of software depends not only the presence of a fault, but also on the probability that an existing error will affect the output, and for safety the potential damage that can be caused as a result of the fault.

A conceptual model of the software reliability process is discussed by Littlewood [Litt85]. The model considers a program p as a mapping from an input space I to an output space O (i.e., $p: I \rightarrow O$). A failure is detected whenever the output resulting from p for a particular input violates the specification. The totality of all the inputs which the program is unable to process correctly is considered as the subset I_F . The central assumption in most of the existing software models is that I_F is encountered purely randomly when the program runs in its use environment. A further source of *randomness* in the failure process occurs when the fault which results in a failure is fixed – hence modifying I_F . The raw data which is used with software reliability models, will be a sequence of execution times, t_1, t_2, \dots, t_{i-1} between successive failures. The objective being to use the data to make intelligent predictions about the future behaviour (in terms of failures) of the software. Such models are often called reliability growth models.

One of the first reliability growth models specifically designed for software was the Jelinski–Moranda model [Jeli72]. However, for most systems predictions given by the

model are considered optimistic, a modified version is discussed by Littlewood [Litt85]. The first model to represent both sources of error was the Littlewood–Verral model [Litt73]. Other models include Duane [Crow77] and Keiller–Littlewood [Keil83]. A major problem with the use of reliability growth models for software is that different models can give greatly varying results; and it is very difficult to predict a priori which model is most suitable for a given software system. The confidence that can be placed in such models is further weakened by the difficulties of data collection.

Other software reliability model classes include: *failure count models*, *fault seeding models* and *input domain base models*. Several models in each class are discussed in a survey paper by Goel [Goel85].

For systems which employ design diversity, several statistical models have been proposed for the attained software reliability [Abde86, Cha86, Scot87, Arla88, Pucc90]. However, none have been developed to the point at which there is consensus on their practicability.

The lack of satisfactory methods to develop and certify computer controlled safety–critical systems, together with the (economical and political) pressures to introduce digital computers into such systems has lead to an urgent need for novel techniques which can be used to develop and certify computer based safety–critical systems.

1.2. Safety

In this section, I will introduce some preliminary (informal) definitions of terms which are often used in the discussion of safety–critical systems. To define the terms, informal definitions of a system and environment are presented. A *system* is the total sum of all its component parts working together within a given environment to achieve a given purpose or mission within a given time over a given life span [Ridl83]. The *environment* within which a system works is all the elements of the universe with which the system interacts. Before considering safety issues in more detail an informal definition of safety is presented [Rodg71].

Definition: Safety

The confidence that the environment that personnel or major items are subjected to is free from inadvertent or unexpected events which may result in injury to personnel or damage to the items exposed.

The safety terms that will be defined are a *disaster*, *mishap* and *hazard* [Youn82]. It should be stressed that it is not being suggested that the definitions are the best possible definitions, rather that they clarify several issues of interest in safety-critical systems.

The first term that I will discuss is that of a *disaster*; this is central to the notion of safety-critical systems, for it is the possibility of a disaster associated with the mission or environment of a system that makes the system safety-critical.

Definition: Disaster

A state of a system constitutes a disaster if and only if the presence of the system in the state means that loss of life, limb, significant revenue or damage to the environment has occurred, during the lifetime of the system.

For a system to be acceptable to a regulatory body (see later), it must be demonstrated that the probability for a system to enter a state which constitutes a disaster is negligible. The above definition clearly specifies the undesired states of a system, this allows us to restate safety in terms of disaster.

Definition: Safety

The confidence that the system does not enter into a disaster.

The term *mishap* is used to denote an unplanned event or series of events (which are caused by the system or its environment) that result in loss of life, limb, significant revenue or damage to the environment, during the lifetime of the system. Mishaps, for a given system, can be simply defined with reference to the disasters of the system.

Definition: Mishap

A mishap is a sequence of changes in the state of a system (which are caused by the system or environment) that result in a disaster of the system.

If attempts are made to design systems which are mishap-free (i.e., the set of states in which the system resides during its lifetime are disaster-free), difficulties will arise from the fact that mishaps (as defined above) involve external environmental impacts on the system, over which the designer may have no control. Hence a stronger notion of safety is required, which removes the involvement of external environmental impacts.

Definition: Hazard

A hazard is a system condition that can lead to a disaster under certain environmental circumstances.

Our basic concern is to ensure that disasters do not occur, but we encountered the problem that the system cannot directly influence all factors that could lead to a disaster. To deal with this problem we introduce hazards as circumstances from which disasters might ensue, but *can* be prevented by the system. We introduce the concept of the *hazard set* of a system as the set of all hazards of that system. Hence, for a given system the absence of hazards (from its hazard set) during the lifetime of the system implies the absence of disasters. The notion of a hazard can be used to define a hazardous event.

Definition: Hazardous event

A hazardous event is a sequence of changes in the state of a system that result in a hazard of the system.

The goal of safety design, stated in terms of hazards, is to design systems that are hazard free (hazardous event free). The main advantage of building systems that avoid hazards rather than disasters, is that all the factors which affect the former are internal to the system.

1.2.1. System Safety

The importance of system safety, as the starting point of an engineering approach to safety, is discussed in some detail by Leveson [Leve86]. An overview of system safety is given in the following paragraph.

System safety became a concern in the late 1940s and was defined as a separate discipline in the late 1950s [Rodg71, Rola83]. A major impetus was that the missile systems

developed in the 1950s and early 1960s required a new approach to controlling hazards associated with weapon systems [Rola83]. The “Minute Man” ICBM was one of the first systems to have a disciplined formal safety programme associated with it. NASA soon recognised the need to have system safety as part of their development programmes. Eventually, the programs pioneered by the military and NASA were adopted by commercial industry in such areas as nuclear power, oil refining, mass transportation, and chemicals. The main contribution of the early safety programmes was the realization that a *system safety philosophy* was required to tackle the development of complex safety-critical systems. A possible definition for system safety philosophy is: “The science which investigates the facts by logic and knowledge to assure personnel and equipment operate harmoniously in a defined environment which will not encounter unexpected or inadvertent events that would result in injury or damage to personnel or equipment.” [Rola83].

System safety engineering is that part of system engineering which identifies all the hazards associated with the system or product and ensures through design and procedures, that these hazards are minimized or controlled. One other area of responsibility which system safety engineering must be concerned with, is the explicit documentation required to protect a company against public liability suits.

1.2.2. Regulation and Legislation

A key property of (conventional) safety-critical systems or hazardous products is that they have been regulated in a legal framework. One of the most regulated industries is the chemical industry where regulations have developed since 1922. In the United States, the most regulated country, there are fourteen main acts which legislate the use of chemical pollutants. Of these acts, the TSCA (Toxic Substances and Control Act) is all-embracing, covering the regulation of both existing and new chemicals. Its implications are far-reaching and it requires industry to furnish the EPA (Environmental Protection Agency) with both technical and business information about production, distribution, use, health risks and the like in relation to the manufacture of any chemical. The TSCA illustrates the basic principle behind legislation for safety, in that it necessitates the

identification of risks related to a legislated product and the provision of data, that demonstrates that the risks have been minimised, in a format that can be reviewed by a legislative agency.

For complex safety-critical systems the main impetus for regulation has come from defence authorities. Of the military standards, one which has been widely used as a basis for the development of safety requirements is Mil-Std-882 [Rodg71]. The purpose of the standard is to provide uniform requirements and criteria for establishing and implementing system safety programmes and to provide guidelines for preparing System Safety Programme Plans. The standard provides clear definitions of safety-related terms, gives both general and detailed requirements for a safety programme and gives a detailed account of the validation of the systems (primarily based on testing).

The need for the specific regulation of safety-critical computer based systems has only recently been recognised. In the United Kingdom the interest in a regulatory framework was fuelled by an ACARD report [ACAR86]. In an appendix of the report entitled “Safety-Critical Software” a possible framework is suggested. The framework is based on a formalised system of registration, certification and licensing of: systems, key personnel and participating organisations. The approach is discussed in detail in a survey paper by Barnes [Barn89], who also discusses frameworks proposed in response to the ACARD report by the Institute of Electrical Engineers (IEE), the Health and Safety Executive (HSE) and US Department of Defense (DOD). The main conclusion of the paper is that the work towards regulatory standards will help to alleviate inconsistencies in the development and assessment of computing systems.

1.3. Process Control Systems

The general class of systems considered in this thesis are *process control systems* [Smit72]. In general, the term process control system covers a large class of systems, which include blast furnaces, petro-chemical plants and avionic systems. I will consider only those process control systems in which there is a computing system. Though each application area has its unique problems, process control systems have significant properties in common. In particular, there is a general relationship between the main

components of a process control system; this relationship is illustrated by the block diagram in figure 1.1, the components are discussed in the following paragraphs.

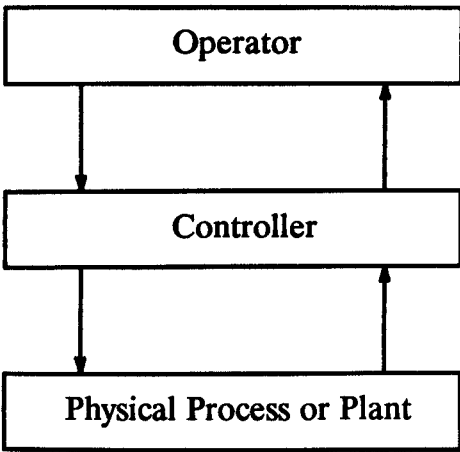


Figure 1.1. Process Control System Components

1.3.1. Operator

The operator, of a process control system, consists of the man (or team of men) which *directly interact* with the controller. The term *directly interact* is used to emphasize that the operator can affect the state of the physical process (via the controller) and react to changes in the state of the physical process. The relationship between the process and the operator can vary from a loosely coupled relationship (e.g., between the operator of a chemical plant and the chemical process) to a tightly coupled relationship (e.g., between the pilot of an aircraft and the aircraft).

1.3.2. Physical Process

The physical process, of a process control system, will be controlled by the process control system. Since the physical process is a physical system its behaviour will be governed (and restricted) by physical laws. As stated previously, the process can vary from a very simple system, such as a railway gate system, to a complex chemical plant. The key observation is that in both cases the behaviour of the process is governed by physical laws. In the case of the railway crossing system physical laws are required to model the

mechanics of a train, and for a chemical plant physical laws are required to model the chemical kinetics of the reactions.

1.3.3. Controller

The controller, of a process control system, is constructed to control the behaviour of the physical process. Typically a controller consists of a control system and a number of sensors, actuators, buttons (selectors) and indicators. The controller interacts with the operator through buttons and indicators; and interacts with the physical process through sensors and actuators.

1.3.4. Controller Components

In the construction of process control systems, the computing systems considered are those that are used in the implementation of the controller. In this section I will discuss the main components of the controller and the role of computing systems in the controller. The relationship between the main components of the controller is illustrated by the block diagram in figure 1.2.

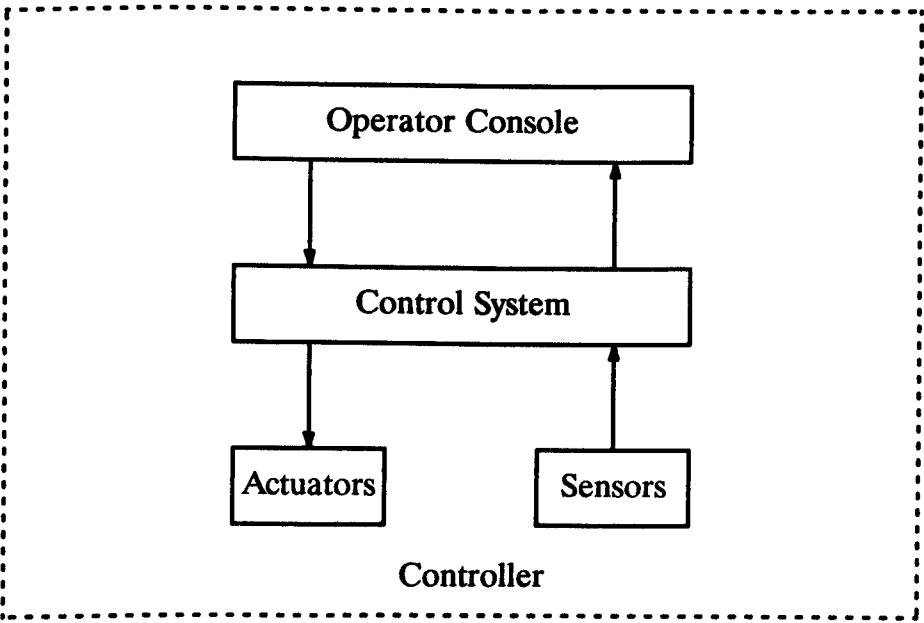


Figure 1.2. Controller Components

Operator Console

The operator console is the means by which the operator interacts with the controller. The operator console consists of dials and indicators. The complexity of the console can

range from a simple display of a few buttons and lights to a complex display produced by computers. The buttons are the devices by which the operator inputs information to the control system and the indicators are the devices by which the control system outputs information to the operator.

Actuators and Sensors

The actuators and sensors of a controller are the physical devices by which the controller influences (via the actuators) and monitors (via the sensors) the behaviour of the process. The actuators can be simple devices which directly control the values of physical properties of the process (e.g., a valve which controls the rate of flow of liquid into a vessel), or complex automatic control devices (e.g., a thermostatically controlled regulator, that ensures that the temperature at a point in the process lies within a particular range). The sensors can be simple devices which sense the values of physical properties of the process (e.g., a thermometer which detects the temperature at a point in the process) and transmit the values to the control system; or the sensors may process the values (e.g., a sensor may compute the rate of change in a temperature) before they are transmitted to the control system. Collectively the actuators and sensors will be referred to as the transducers of the system.

Control System

The control system of a controller is the component which uses the actuators and sensors to control the behaviour, of the physical process. In particular, the control system must ensure that the physical process exhibits the required behaviour.

Interfaces

To specify the behaviour required to be exhibited by the controller, the interface between the controller and the controlled environment must be delineated. The interfaces are illustrated by the block diagram in figure 1.3.

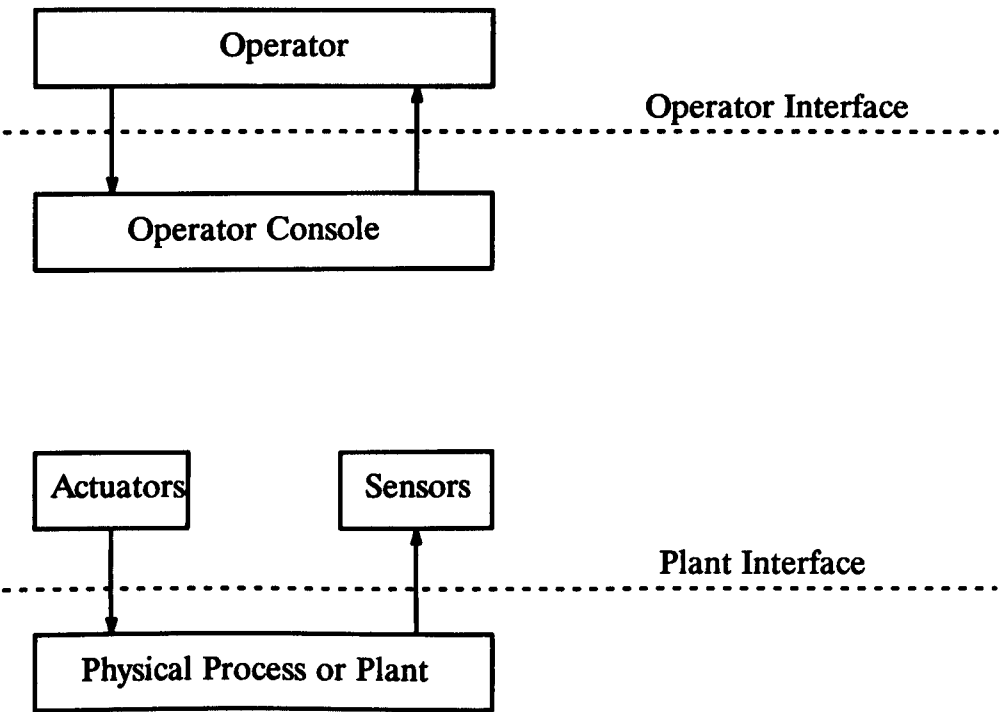


Figure 1.3. Controller Interfaces

In the block diagram of the controller two interfaces to the control system are illustrated: i) the *operator interface*, the interface between the operator console and the operator and ii) the *plant interface*, the interface between the transducers (i.e., actuators and sensors) and the plant. It is through the operator interface that the operator interacts with the controller. The operator can view the status of the controller through the indicators and modify the status of the controller through the buttons. It is through this plant interface that the controller interacts with the physical process. The controller monitors the behaviour of the plant through the sensors and modifies the behaviour through actuators.

1.4. Safety in Computer Based Systems

In this section, a brief discussion of how safety issues have been treated in computing systems is given. Also the class of computing systems which will be considered in this thesis are precisely defined. Recently, safety in computer based systems has been closely related to the attributes of reliability and security. Some researchers feel that these attributes of safety, security and reliability have qualities which are so similar that they are facets of a

single attribute – this generic attribute is referred to as dependability [Lapr85]. We will consider dependability as being the generic term which means the extent to which the behaviour of a system conforms to a requirement R. Clearly safety and reliability are special cases of dependability, safety being the case when the requirement R is the “safety requirement” of the system, reliability the case when the requirement R is the “mission requirements” of the system; and security the case when R is the “security requirement”. In safety-critical applications the main dependability goals for the system are: safety and reliability. It should be recognised that a reliable system is not automatically a safe system. A detailed analysis [Leve83a], has shown that though high reliability, may be necessary it is not sufficient to ensure safety. It is therefore important to distinguish between reliability and safety. A more detailed discussion of why safety and reliability should be considered as separate issues is given later. In the following subsections reliability and safety are discussed in the context of computing systems.

1.4.1. Reliability

Reliability in computing systems has been investigated for many years [Ande79]. This has lead to a situation in which reliability is a well understood concept, and most researchers and practitioners would agree that it should be defined as a quantitative attribute (more specifically a probability). One such definition is given below.

Definition: Reliability

The reliability of a system is the probability of accomplishment of a function under specified environmental conditions over a specified time [Mulz85].

It should be clear from the (above) definition that reliability is oriented towards the purpose of a system and to the intended action, it is the extent to which a system can be expected to perform the specified task.

1.4.2. Safety

Research into safety in computing systems is relatively (to reliability research) recent. Major research started in the late 1970s [Safe79]. A consensus on a suitable technical definition of safety has not been reached. However, if we are willing to consider safety as

an aspect of dependability then the reliability definition can be simply modified to a suitable definition for safety.

Definition: Safety

The safety of a system is the probability the system will satisfy the safety requirements under specified environmental conditions over a specified time.

The safety requirements of a system express the conditions that the behaviour of the system must satisfy for the system to remain free of disasters.

1.4.3. Safety-critical Computing Systems

A precise definition of the class of safety-critical computing system which will be considered in this thesis, is given in the context of process control systems.

Definition: Safety-critical computing system

A safety-critical computing system is a system that is a component of a process control system which can cause or allow the overall system to enter into a hazardous state.

It should be noted that in the definition there is no reference to a failure (in the mission), or any mission related behaviour. The definition is given solely in terms of hazards, that is the possibility of disasters.

1.5. Obstacles to Computers in Safety-Critical Systems

In this section, I will discuss some of the obstacles to the introduction of computing systems into safety-critical systems. The discussion is given in terms of the confidence that can be placed in the safety of the safety-critical system. The confidence that a designer or organization has in the safety of a system is rarely based on one piece (or type) of evidence, rather on a collection of “facts” which indicate that confidence can be placed in the safety of a system. The type of evidence can vary from rigorous scientific experiments, formal mathematical proofs to comparisons with similar systems. In the following paragraphs, I will discuss some of the reasons why greater confidence can be placed in the safety of conventional safety-critical systems than in safety-critical computing systems. The reasons will be placed into two broad categories intrinsic and application dependent.

1.5.1. Intrinsic Obstacles

In this section I will discuss five obstacles to the introduction of computers into safety-critical systems, which arise from the the differences between the intrinsic nature of computing systems and conventional (non-computer) systems.

Complex Mathematical Models

Most properties of conventional systems can be modelled and analysed using continuous mathematics. The main advantage of these models is that they are much easier to understand (and analyse) than the actual systems. There are some conventional systems which cannot be modelled using continuous mathematics, such as relay systems, but for these systems computational tools for analysis and synthesis are available. However, most properties of safety-critical computing systems are modelled by discrete mathematics. The main disadvantage of having to use discrete mathematics is that the complexity of the models for discrete systems is about the same as that of the actual system. The main reason which is usually given for the difference in the effectiveness of the models is that discrete mathematics is less mature than continuous mathematics. Although this is true to some extent, the inherent complexity of the computing systems being analysed will probably always lead to complex mathematical models. Hence, some means to handle this complexity is necessary.

Maintenance

Maintenance of conventional systems is usually straightforward, the main reason for this is that the systems are loosely-coupled (i.e., relative to computing systems) which allows maintenance engineers to limit the influence of their actions. It should be a worrying fact that despite the loosely coupled nature of conventional systems many disasters are caused by maintenance actions, in some cases even when the maintenance was meant to enhance safety [Reas87]. Unless carefully planned, the maintenance of conventional systems can cause problems in seemingly unrelated areas. Problems in the maintenance of computing systems arise since the components of computing systems are usually tightly coupled. In a tightly coupled, complex system, the consequences of maintenance actions radiate like ripples in a pool, but too often maintenance engineers

see their influence only within the narrow sector of their current concern. Given the fact that maintenance actions have caused many disasters in conventional systems, and the problems of software maintenance [Somm82], the maintenance problem must be of grave concern to the designers of safety-critical computing systems.

Limited Predictability

Generally speaking, conventional systems are built from standard components, for which good quality data on failure rates and failure modes is available. The faults which are of major concern in conventional systems are random wear and tear faults in the components, and not design faults. It is assumed that most design faults have been eliminated by a careful systematic design methodology. Note this assumption does not preclude the possibility of design faults it merely states the belief of conventional designers, that the contribution of design faults to the unreliability of a system is negligible to the overall unreliability of the system. (However, with the increased complexity of conventional systems, this assumption may no longer be justifiable.) The reliability data of the components (together with the assumption that design faults are insignificant) allows designers to predict reliability values for the system.

Safety-critical computing systems are rarely built from standard components (e.g., bespoke software packages). Though some software packages are available, even for these packages the data on failure rates and failure modes is limited by the level of reuse. The faults which are of major concern in computing systems are design faults. Hence there are two obstacles to accurate reliability predictions of computing systems, firstly only limited reliability information is available on the components of the system, and secondly faults in the design that specifies the interactions between the components can have a significant effect on the reliability. Though reliability models have been built to predict the reliability of software systems generally only limited confidence can be placed in their predictions. This is largely due to the fact that information on the reliability of the components which must be used for the parameters of the models is limited to that gained from testing during the development of the software [Pucc90]. Furthermore since it is clearly inappropriate for the system to be tested in the final execution environment – the system must be tested

in a simulated test environment. Hence the quality of the data collected during such testing is limited by the accuracy in which the probability distribution of the input space of the simulated environment reflects that of the final execution environment. However for conventional systems the components are usually standard components for which good quality reliability data is available, hence the parameters for the reliability models of conventional systems can be stated (based on good quality historical data) with a high degree of confidence.

Utilization of Redundancy

When redundancy is employed in conventional systems, the fault hypothesis adopted is that the occurrence of faults in redundant components is independent. Justification for using this fault hypothesis is based on the nature of the faults which are considered to be most significant in conventional systems. Under the independence of faults assumption the probability of mutual failure can be computed as the product of component failure (i.e., $\text{Prob}(A \text{ and } B \text{ fail}) = \text{Prob}(A \text{ fails}) \times \text{Prob}(B \text{ fails})$). This allows redundancy to be used as a powerful tool in the design of conventional systems. From the nature of faults in software systems redundancy based on simple replication is of little use. A more useful form of redundancy in computing systems is design diversity, that is, redundancy of design to provide tolerance to human mistakes made in the design of the computing system. The effect of common mode failures in diverse designs is difficult to evaluate. Experiments have shown that common faults between different designs are likely to exist, though they have also shown that redundancy can be useful [Bish86, Sagl86, Ande85]. However, there is still a question mark over the true effectiveness of redundancy, and it is probably highly-dependent on the nature of the system.

Safety Factors

The safety-critical requirements of (many) conventional systems are continuous and this allows designers to add tolerance factors to the requirements of the components. For example, suppose to avoid a hazard a component (vessel) in a system must be capable of withstanding pressures up to 300 kilopascals – the designer can introduce tolerance by using a vessel capable of withstanding 400 kilopascals. The discrete nature of the

safety-requirements of computing systems makes the introduction of tolerance at a component level more complicated. With the possible exception of timing requirements, to introduce tolerance into the realisation of computing systems, analysis within a system context is required.

1.5.2. Application Obstacles

In this section, I will discuss six obstacles to the introduction of computers into safety-critical systems which arise from the nature of the application areas into which the computers are being introduced.

Limited Experience

The application areas into which safety-critical computing systems are being introduced are almost always novel application areas for computing systems, hence the designers of such systems have limited experience in the construction of the systems. This lack of experience can lead to problems in communication between the members of the development team, and confusion over the roles of the members in the team [Leve89]. The problems in communication are most acute during the requirements analysis (i.e., before an overall picture of the system is available). The communication problems are of particular concern when it comes to safety issues. For in development teams, with weak communication between the members, it is possible for all the members to think that safety is some other members responsibility.

Application Complexity

The application areas into which safety-critical computing systems are being introduced have complex operator and plant interfaces. The complexity in the interfaces makes the task of assessing the effect of the computing systems behaviour on the controller (hence overall system) very difficult. Further problems caused by complexity of the systems are that the requirements specifications of such safety-critical computing systems tend to be complex leading to problems of incompleteness and inconsistency.

Complex Operator Interface

The problems caused by complex operator interfaces are of particular importance in safety-critical systems in light of the role that humans play in mishaps [Vend80, Rous81,

Perr84]. For application areas into which computing systems are being introduced, the operator interface is usually complex, and sometimes there is dynamic allocation of tasks between the operators and controlling system. This means that human factors must be considered which can lead to complicated issues of ergonomics and cognitive psychology.

Manual Intervention

Most of the application areas of conventional systems have had both low complexity and slow time constants; this has allowed the possibility of manual intervention in the avoidance of disasters. However, the application areas in which computing systems are being introduced have high complexity and fast time constants making manual intervention very difficult. Even in situations where manual intervention is possible, operator complacency (due to long periods of inactivity) make it very risky to depend on such intervention for safety [Leve86].

Global Problem

It is well recognised and often stated that safety is a global property, that is, safety is a property of a system in a environment, and can only be assessed in the context of the environment [Gors86, Kell86, Reev86]. Hence global tools are required to support the analysis and design of safety-critical systems. However, there are no global tools available for the design and analysis of computer based systems [Leve89].

Extreme Demands

Though computers may be able to improve the overall safety of a process control system by replacing the conventional controller by a computing system, in practice it may be difficult to realise the improvement in safety because of the extreme demands imposed on the functions of the overall system. The previous point is probably true for all improvements in technology, for example Perrow [Perr84] argues: “that although technological improvements reduce the possibility of aircraft accidents substantially, they also enable those making decisions to run greater risks in search of increased performance.” The key observation being that as the technology improves, the increased safety potential is not fully realized, because of increased demands on efficiency and functions.

1.6. System Lifecycle

The phases of the system lifecycle and the relationships (in time) between them are crudely illustrated by the block diagram in figure 1.4.

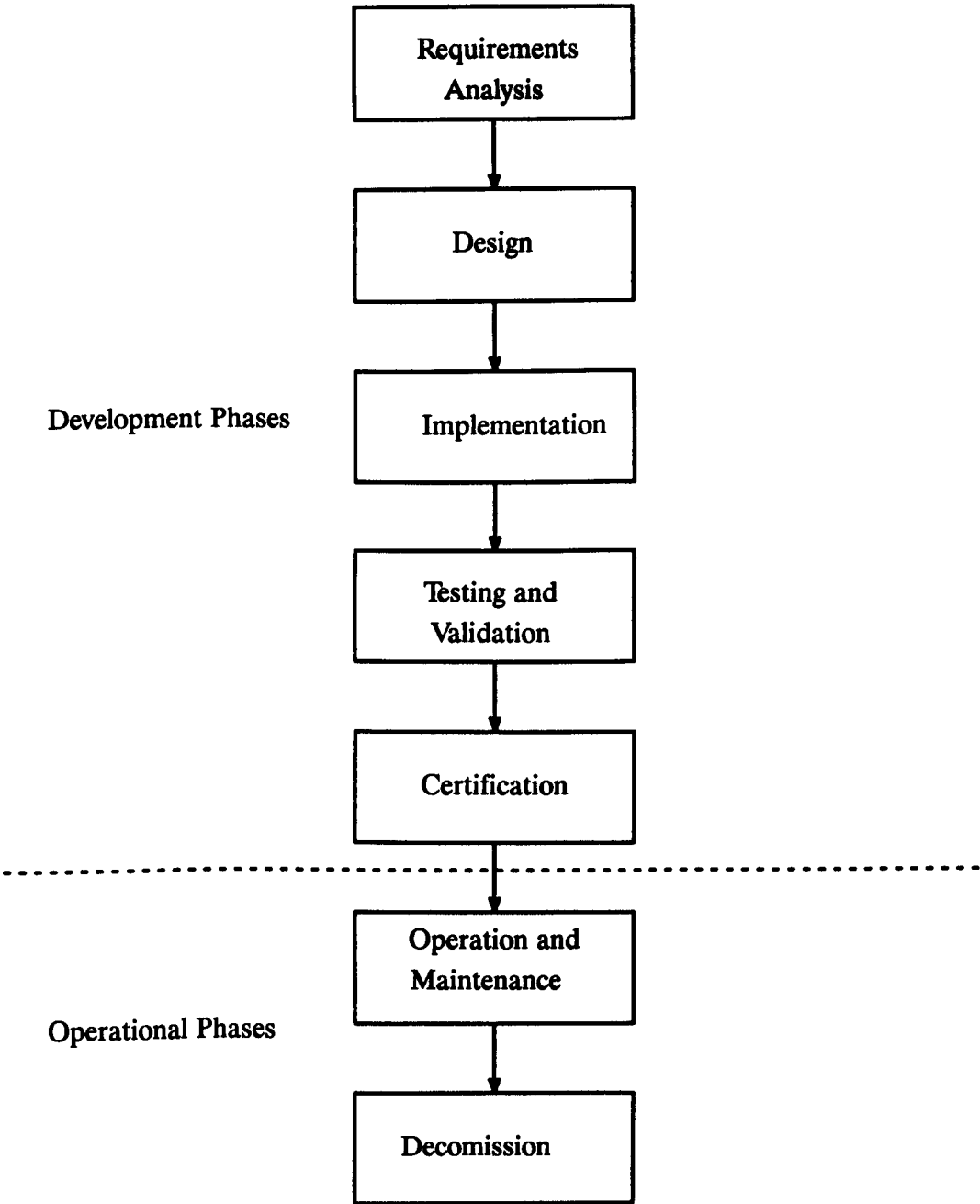


Figure 1.4. Simplified System Life Cycle

The development life cycle is represented by the first five phases, and the operational life cycle by the last two phases. In the (minimal) model development is shown to progress in one direction, however, in the development of most systems backtracking between the

development phases will be necessary. These phases are briefly discussed in the following paragraphs.

Requirements Analysis

The requirements analysis phase is the first phase of development, and as such at the start of the phase there is only an informal representation of intended system behaviour. During the requirements analysis the intended system behaviour must be analysed, so as to construct a formal representation (the benefits of a formal representation will be discussed later) of the intended behaviour on which an agreement can be reached amongst all the interested parties. Since safety can only be assessed in the context of an environment, the requirements specification must also give a specification of the behaviour of the environment into which the system will be installed. Good requirements analysis should allow the resolution of the complexities in the problem domain (by modelling the interactions between the requirements), before the introduction of complexities from a solution to the problem. If possible, the requirements should be certified by the appropriate licensing authorities.

Design

During the design of a safety-critical computing system the tasks (which the system must perform to satisfy the requirements) will be partitioned between the hardware and software of the computing system. For the software subsystem, the design can be given in terms of the behaviour which can be observed at the interfaces between the software components. Further issues, that the design must address are what techniques should be used to provide design redundancy, suitable techniques include N-modular redundancy [Ande81], Recovery Block schemes[Scot83] and Deadline schemes [Camp79], and in which areas of the design the redundancy is best employed. The design phase is linked to the computing system requirements by a development process which refines the requirements to construct the design, and a verification process which must ensure that the design specification complies with the requirements specification.

Implementation

The implementation of a safety-critical computing system is the construction of the components introduced in the design specification. The implementation phase is linked to the design by a development process which refines the design specifications of the components, and a verification process which must ensure that the implementation complies with the design.

Validation

The criteria for the validation of a safety-critical computing system must be derived from the requirements specification. Two different, but not exclusive, approaches can be used for the validation of the system: formal verification and test based strategies. During this phase of the system an evaluation of the safety of the system must be performed. In particular the critical components of the system must be identified; and verification and testing criteria be formulated for the system. The criteria chosen must be stringent enough to satisfy the licensing authorities, yet simple enough to allow confidence to be placed in the results of the validation.

Certification

The development of safety-critical systems is usually controlled by regulatory authorities. These authorities impose certification criteria, in accordance with the safety standards established for different sectors, which must be satisfied before the system can be put into service. Essentially, for a system to satisfy certification criteria it must be possible to provide evidence (usually in a specified format) to support the claim that the system will meet the imposed safety standards. The need for certification of such systems brings the issue of assessment to the forefront, hence special emphases must be placed on the collection of evidence and development of models to allow accurate predication of the achieved level of safety in a system.

The limits to a numerical assessment of software using the current technology (discussed earlier), has meant that current practice in software quality assurance concentrates on the software development process. This involves agreeing the procedures necessary to complete the individual processes involved in each stage of software

development and checking that each process is completed in accordance with the agreed procedures. In the certification phase the regulatory authorities must check the evidence produced during the development of the system, to confirm that the system can be put into service.

Operation and Maintenance

The operation and maintenance phase of the system should be the longest phase of the system life cycle. Any modifications which must be performed during this phase must be carefully considered. In particular the behaviour of the overall system, as expressed by the requirements specification must be considered. Two important cases are modifications in the requirements of a system, and changes in the environment of the system. If the maintenance is in response to a modification of the requirements of the system, the requirements specification must be modified to assess the consequences of the modification on the critical behaviour of the system. If any changes are proposed in the environment of the system, the model of the environment in the requirements specification must be modified to allow an analysis of the consequences of the proposed changes.

Decomission

In this phase the computing system is decomissioned. In safety-critical systems care must be taken that the behaviour of the system during the decomission phase will be free of disasters. Furthermore, the experience gained during the lifetime of the system should be carefully documented – to guide the design of a similar system at a later date.

1.7. Aims

Of the seven phases I will concentrate on requirements analysis, in the rest of this thesis. The primary aim of the thesis is to provide a formal basis for the requirements analysis of safety-critical computing systems. The basic strategy used to achieve this aim is to provide a formal specification model which can specify all the relevant (logical and timing) properties of the system in a structured format, and a development model which outlines the main phases of requirements analysis.

This thesis concentrates on the provision of a framework for requirements analysis of a specific class of systems – process control systems. The thesis shows how the adoption

of a framework can provide a means to structure the specifications produced during the requirements analysis. This structure enables a formal link to be established between the different levels of specifications, and allows the derivation of general verification techniques. Thus providing a means to re-use the basic theory that links the specifications.

To demonstrate how the approach can be used for the requirements analysis of safety-critical computing systems, a working example is presented with the development of the approach; and a full example is given in the appendices B and C.

1.8. Overview

In chapter two a detailed discussion of the requirements analysis is presented. The major obstacles which arise during requirements analysis are highlighted and the role of a requirements specification in system development is discussed. A structured approach to requirements analysis is introduced. The approach consists of three parts: a system development model, a formal model and a related development methodology. The development model is discussed in some detail. Finally, a simple chemical plant is described, which will be used as a worked example to illustrate how the development model and formal model can support requirements analysis.

Chapter three introduces the basic concepts behind the formal model. These will include the formalisation of the time base and the system state, and the introduction of the semantics of the model in terms of formal system *histories*. Formal structures which can be used to specify the behaviour of safety-critical computing systems are presented and several examples are given.

Chapter four further develops the constructs introduced in chapter three, to formulate a high-level specification construct – a *mode*. It is shown, by examples, how modes can be used to concisely represent system behaviour during an interval of time. The formal properties of the modes are discussed. This chapter also describes a construct which can be used to compose modes in a structured format – a *mode graph*. Finally, some formal tools which can be used to analyse specifications constructed using mode graphs are discussed.

Chapter five deals with how to express the specifications produced during the real world analysis, in terms of the formal constructs introduced in chapters three and four. Specific properties of the formal constructs of the real world specifications are discussed, and some example specifications presented. Similarly, chapter six, deals with how to express the specifications produced during the controller analysis, in terms of the formal constructs introduced in chapter four.

Chapter seven presents systematic methodologies for the development of the real world specifications (discussed in chapter five). The methodologies presented in this chapter deal with how best to integrate traditional system safety techniques into the general approach to requirements analysis.

Chapter eight presents systematic methodologies for the development of the controller specifications (discussed in chapter six). The controller specification are constructed by an analysis of the real world specifications. Since the real world and controller specifications are specified in the formal framework, verification strategies form an integral part of the methodologies.

The final chapter presents some conclusions from the work presented in this thesis, discusses how far the aims of the thesis have been achieved and gives some suggestions for future work.

Chapter 2 - Requirements Analysis

2.1. Introduction

Requirements analysis plays a vital role in the development of safety-critical systems, since any errors in the identified requirements will corrupt the subsequent stages of system development. Experience in safety-critical systems has shown that errors in requirements specification are one of the major causes of mishaps [Leve86].

Five reasons why the problems of requirements analysis are so acute are outlined below, they will be discussed in more detail during the chapter.

- The elicitation of the requirements often needs to be performed by a *multidisciplinary team*. Unless good communication channels and clearly defined roles and responsibilities are associated with the team members, problems in communication and confusion over the roles of the team members can lead to poorly constructed requirements.
- The fact that safety is a *global issue* can causes major problems during the requirements analysis, since no overall picture of system behaviour may be available at the start of the analysis.
- If the safety and mission issues of a system are *intertwined* during the analysis, the elicitation of both types of requirements are complicated.
- For many systems, it can be difficult to decide the *levels of abstraction* at which the analysis should be performed.
- The (logical and timing) requirements of the system and the behaviour of the environment are usually represented in *different formal frameworks*; this can complicate the analysis of total system behaviour.

Current methods for requirements elicitation for safety-critical systems are based on a combination of system safety and software development techniques, which attempt to overcome the five problems outlined above. An approach is the use of HAZOPS (Hazard and Operability Studies) and FTA (Fault Tree Analysis) to identify hazards from which the software safety requirements are produced [Leve89]. General guidelines for the integration of system safety and software development techniques are available [Barn89,

HSE87]. However, the main disadvantage of the current methods, which can lead to weak (poor) requirements, is that no unified system wide approach for requirements analysis is available.

There are few formal methods oriented towards requirements specifications, however the work of the Alvey FOREST project [Maib87, Pott86] developed a formalism for representing requirements and a method for requirements capture. The formalism is an extension of modal logic – Modal Action Logic (MAL). The method is based on structured methods such as data flow analysis and entity relationship analysis, and incremental formalization – structured common sense (SCS). Though this project tackled issues similar to those considered in this thesis, there are two significant differences. Firstly the FOREST project considers a much wider class of systems, this thesis concentrates on process control systems. Secondly, the FOREST project did not specifically deal with safety–critical aspects of systems, an issue which is central to the work presented in this thesis.

2.2. Role of Requirements Specification

The pervasive nature of the requirements specification can be seen from the relationships (indicated by the paths 1 to 5 in figure 2.1) between the requirements specification and the other phases of the system life cycle. The classification of the roles of the requirement specification, is a modification of the roles defined by Wasserman [Wass79] and Gorski [Gors88]. The main modifications are the inclusion of the certification phase and an emphasis on safety–critical issues during the other phases. The relationships are discussed in the following paragraphs.

- 1) The requirements specification is a means of precisely stating the requirements of the customer. The requirements specification is compared against the system concept (requirements definition) to confirm that the behaviour expressed by the requirements specification is indeed the required behaviour of the system.
- 2) The requirements specification provides a structured representation of the problem. The system designers can exploit this representation to guide their design. The requirements specification also provides a formal statement against which different

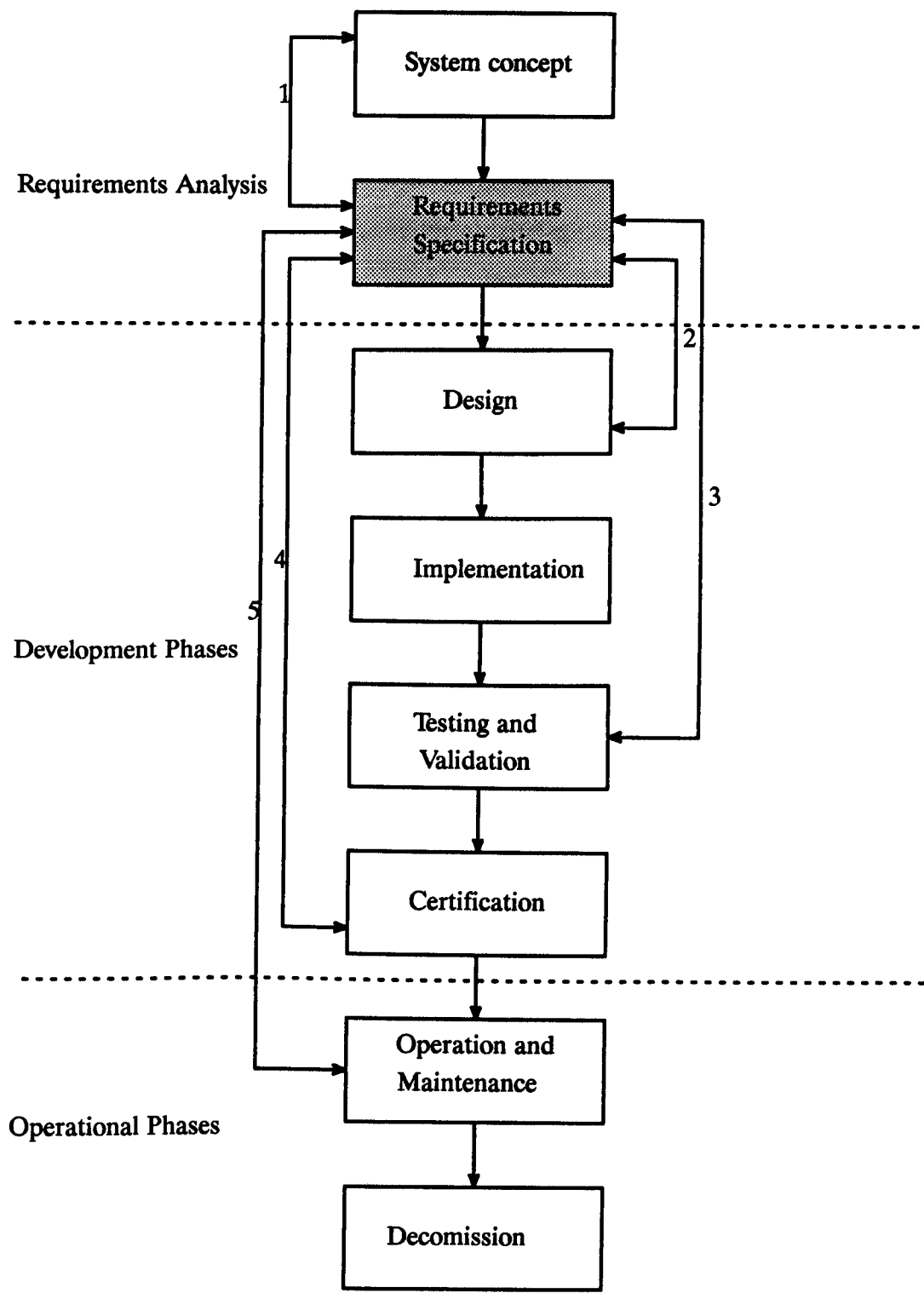


Figure 2.1. Role of Requirements Specification

designs can be verified and evaluated. It is possible that problems in the design may force changes in the requirements specification. In particular, issues related to conflicts between reliability and safety may force the requirements specification to be modified. A detailed discussion of how the requirements specification can resolve conflicts is given later.

3) The requirements specification is the statement of system behaviour against which the implementation can be tested. The requirements specification can therefore be used to generate test criteria for system testing. To be useful in generating the tests the requirement specification must stipulate the required behaviour of the system in an analysable format.

4) In section 1.6.7 it was stated that it is desirable for the requirements specification to be certified by the licensing authorities before the system enters the design phase. The main benefit of a certified requirements specification is that it can help the licensing authority with the certification of the implemented system, by providing the authority with a precise statement of system behaviour – against which the verification condition and tests used in the verification and testing of the system can be checked.

5) The problems which can be caused by modification and maintenance were highlighted in section 1.5. To overcome these problems a thorough understanding of the behaviour of the controller in a system context is required; this can be provided by a clear requirements specification. As a general rule, after the system has been certified any modifications to the system must be preceded by the corresponding modifications in the requirements specification. The consequences of such modifications to the requirements specification must be evaluated before the system is modified.

2.2.1. Requirements Specification Viewpoints

Two different views of the requirements specification can be identified, the customer's view and the developer's view (in some cases the developer's view is partitioned into a design view and verification view [Wass79]).

The *customer's view* of the requirements specification is concerned with the relationship between the requirements specification and system concept. More

specifically, this view must be useful in assuring that the requirements specification accurately reflects the intended requirements of the system. As such the customer's view must be a high-level specification of the requirements expressed in terms of the "real-world" issues of concern to the customer relating to the behaviour of the system and the environment in which the system will be installed. To facilitate good communication during the requirements analysis, the terminology used in the specification should be consistent with the application area[Yeh80]. In terms of the general model of the process control system, the customer's view is concerned with the behaviour exhibited at the operator console and by the physical process.

The *developer's view* of the requirements specification is concerned with the relationship between the requirements specification and the subsequent phases. More specifically the view must give a precise description of the required behaviour of the system in terms of the components of the controller (i.e., the sensors and actuators) and the relationships between the components and the physical properties of the plant. In terms of the general model of process control systems, the developer's view is concerned with the behaviour exhibited at the operator console and the sensors and actuators of the system.

For the two views of the requirements specification to be useful it must be possible to show that both views are consistent. More specifically, it should be possible to formally verify that if system behaviour complies with the developer's view of the requirements specification, then it must comply with the customer's view of the requirements specification.

Analysis of a system, for the construction of the customer's view, is performed over the operator console and the physical properties of the plant. Hence the customer's view analysis will be referred to as *real-world analysis*. Analysis of a system, for the construction of the developer's view, is performed over the operator console and the sensors and actuators. Hence the developer's view analysis will be referred to as *controller analysis*. A similar partition of the requirements analysis is suggested by Bishop [Bish86] who outlines an approach in which invariants are used to specify the behaviour of the system, at two levels.

2.3. Structured Requirements Analysis

A structured approach to the requirements analysis of safety-critical computing systems is proposed. The basic approach consists of three main parts:

- a development model;
- a formal model and
- a development programme.

In the following sections the scheme is discussed in some detail, in terms of the three parts. The discussion will include the benefits gained from each part of the scheme, in particular how the scheme can help to resolve some of the general obstacles to the introduction of computing systems (see section 1.5) which arise in subsequent development stages, and the specific difficulties in requirements analysis.

2.3.1. Development Model

In the structured approach, system behaviour is analysed at two distinct levels of abstraction: *real world* and *controller*. At the real world level the analysis is over the behaviour exhibited by the physical process and at the operator console. The properties of the environment of interest, at the real world level, are the physical laws which govern the behaviour of the physical process, and the properties which arise from the construction of the plant. At the controller level, the analysis is over the behaviour exhibited by sensors and actuators of the controller and at the operator console. The properties of the environment of interest, at the controller level, include the relationships between the sensors, actuators and the physical process. It should be noted that the operator console is in both levels, at the real world level it represents the role of the operator and at the controller level it represents the information passed from and to the operator.

2.3.1.1. Separation of Issues

For safety-critical systems it has been argued that for a clear analysis of the safety-related properties, a distinction must be made between the safety-critical and mission-oriented behaviour of a system [Leve82, Leve84, Mulz85, Redm85]. In the

structured approach, the distinction between the safety and mission issues is established during the requirements analysis. The *safety requirements* of a system are the requirements that the system must satisfy to ensure that it is free of any disaster states during its operation. The *mission requirements* are the requirements that a system must satisfy to fulfil the purpose of building the system. Clearly, if the purpose of the system is to perform a safety function, then a separation cannot be made. Three potential benefits gained by a separation of the safety and mission issues during requirements analysis are discussed below.

Resolution of Conflicts

In the system requirements of safety-critical systems conflicts in the required behaviour of the system may exist between safety and mission-related issues. To allow trade-offs to be made between such conflicts they must be clearly identified, this can be achieved by the separation of the system requirements into the safety and mission requirements. The identified conflicts can then be used to analyse how increased demands on the functions of the mission affect safety, and to identify the restrictions which would have to be placed on the functions of the mission to construct an *intrinsically* safe system (in the sense that any hazards are eliminated by modifications to the mission).

Focus on Safety-Critical Issues

By treating the safety requirements as a separate issue, the safety requirements can be derived, and analysed, before a detailed analysis of the mission requirements is performed. In doing so, a solution independent statement of safety is developed.

Clarification of Certification Criteria

The separation allows a clear distinction to be made between critical and mission failures. This distinction allows the high reliability standards of the licensing bodies (on the order of 10^{-9} failures/hour) to be expressed with respect to critical failures. That is, by standards of the form: at most 10^{-9} critical-failures/hour, where a critical failure is a failure which can lead to a disaster. This form is preferable since it emphasises critical

failures, hence it forces the designer of safety-critical systems to precisely define the notion of a critical-failure. It is hoped that by clarifying the certification standards, using the notion of critical failures, it may be possible to both achieve and demonstrate that systems satisfy the standards required by licensing bodies. This hope is based on the premise that the task of demonstrating that the required standards have been met, will be simplified by the availability of a precise definition of critical failures.

2.3.1.2. Safety-Critical System Structure

A general structure for safety-critical systems that emphasises the separation of safety and mission issues is illustrated by the block diagram in figure 2.2. Separation at the controller level is illustrated by the block diagram in figure 2.3. The safety-critical system structure introduces two subsystems, the safety and mission subsystems. The subsystems and the interaction between them is discussed in the following paragraphs.

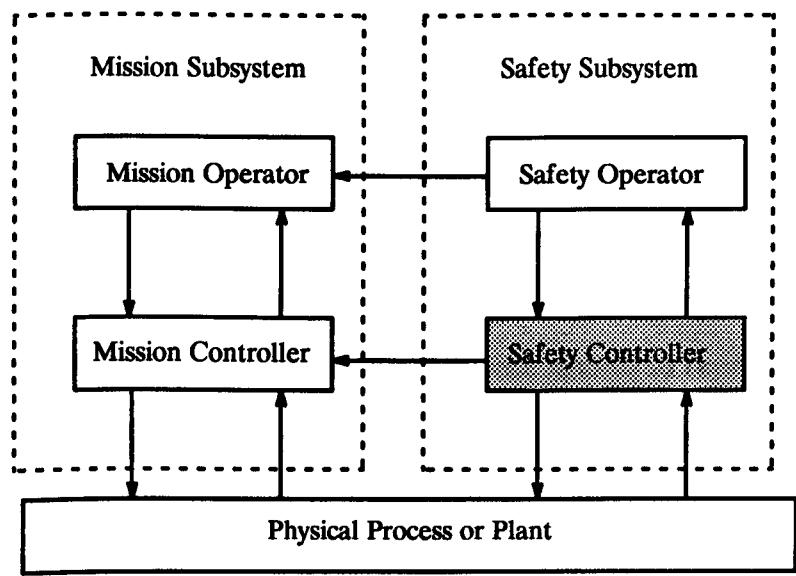


Figure 2.2. Safety-Critical System Structure

Justification for the feasibility of separation of the mission and safety controller arises from an historical perspective and from the nature of safety. Typically, nuclear systems [Parn88] and railway control systems [Theu86a] have been produced with separate systems to handle the safety aspects. Safety is an invariant property, provided that the initial state is not a hazardous state, for safety it is sufficient for the system to ensure that a hazardous

event does not occur. Thus provided sufficient redundancy can be employed, by allowing the safety controller to concentrate on preventing hazards, the safety controller can be separated from the mission controller. However, even if a full separation between the safety and mission controller cannot be realised, considerable benefit can be gained from a partial separation.

Safety Subsystem

The *safety operator* is the operator which interacts with the safety-critical functions of the process control system. The distinction is made between a general operator and safety operator to emphasize the restricted access to the safety controller. In the diagram an arc connects the safety operator to the mission operator; this is used to represent the intention that the safety operator can influence the mission operator (and hence, the mission subsystem).

The absence of an arc from the mission operator to the safety operator represents the intention that the mission operator cannot influence the safety operator. The *safety controller* is the control system which performs all the safety-critical functions of the controller. This controller should be kept as simple as possible, that is the choice of sensors and actuators of the system should be made on the basis of the minimal functionality required to ensure safety. In the block diagram there is an arc which connects the safety controller to the mission controller. The link is used to represent the intention that the safety controller may influence the behaviour of the mission controller. The absence of a link from the mission controller to the safety controller represents the intention that the mission controller cannot influence the safety controller.

Mission Subsystem

The *mission operator* is the operator that interacts with the mission related functions of the process control system. The *mission controller* is the controller which performs the mission related functions of the controller. The sensors and actuators which are used to build this system will typically be more complex than those of the safety controller. Though the mission controller cannot directly influence the safety controller, it may be able to

influence the safety controller indirectly by changing the value of properties in the physical process which are subsequently sensed by the the sensors of the safety controller.

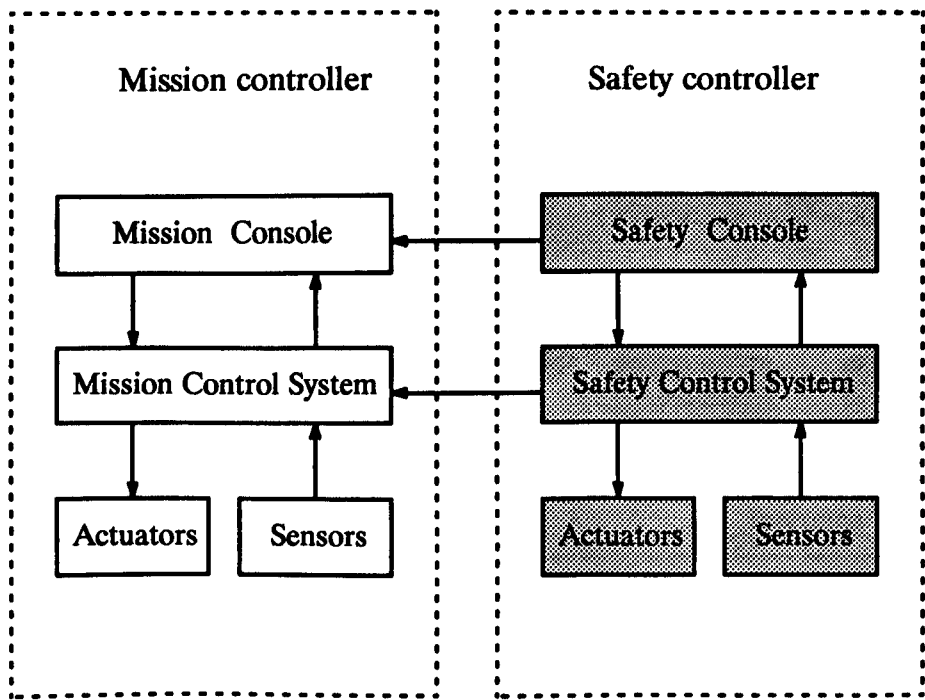


Figure 2.3. Safety-Critical Controller Components

Safety Controller

The components of the *safety console* are physical buttons and physical indicators; the buttons and indicators of the safety console should be kept simple as should the interactions between the safety operator and safety controller. The *safety control system* is the control system that implements the control functions of the safety controller (hence the safety-critical functions of the process control system). The *sensors* and *actuators* are connected to the safety control system. Typically the sensors are used to monitor the state of the physical process (or mission controller) in order to detect the presence of any potentially hazardous states, and the actuators are used to affect the state of the physical process to ensure that the system does not enter a hazardous state. These sensors and actuators must be very reliable (hence they should be kept simple). The two interfaces of the safety controller, the *safety console interface* and *safety plant interface*, are illustrated in the block diagrams in figure 2.4. Both interfaces should be kept as simple as possible.

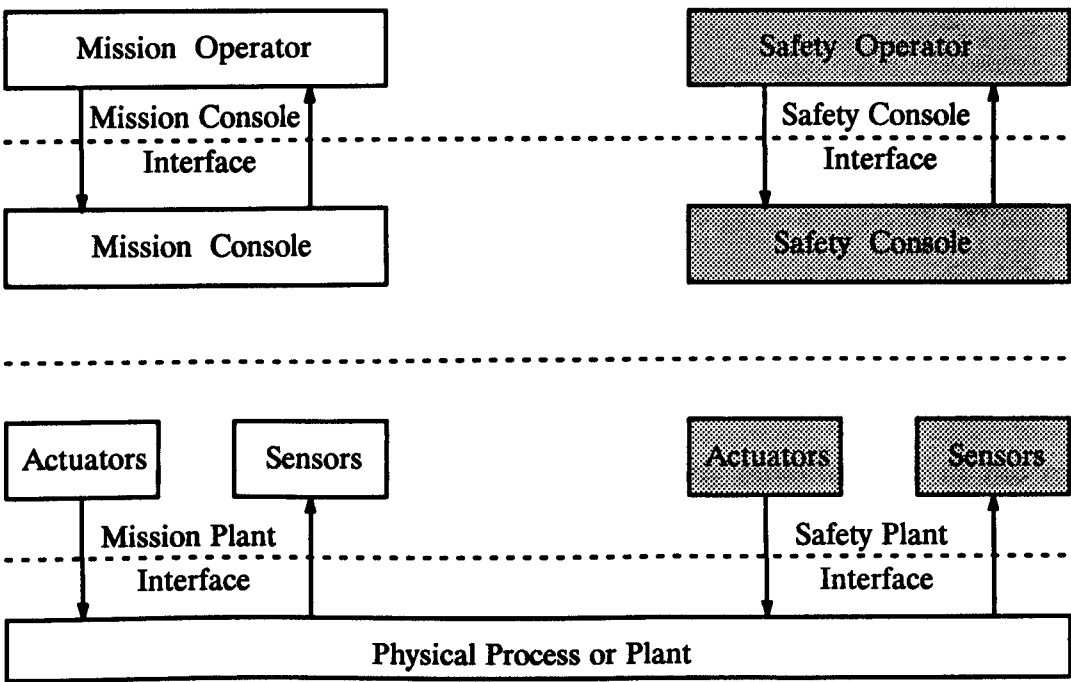


Figure. 2.4. Safety-Critical Controller Interfaces

Mission Controller

The dials and indicators of the *mission console* will typically be more complex than those of the safety operator console. The *mission control system* is the control system that contains the control functions of the mission controller (hence the mission related functions of the process control system). The *sensors* and *actuators* are connected to the mission control system. Typically the sensors will be used to check the state of the physical process and the actuators to modify the state to ensure that the behaviour of the physical process complies with the mission. The sensors and actuators are likely to be more complex than those of the safety subsystem. The two interfaces of the mission controller, the *mission console interface* and *mission plant interface*, are illustrated by the block diagrams in figure 2.4.

Benefits of the General Structure

The general structure for safety-critical systems can help in the resolution of several of the obstacles (to the introduction of computing systems into safety-critical applications)

that arise in the subsequent development stages. These specific areas are discussed in the following paragraphs.

Restrictive Maintenance

Clearly, it is desirable that the maintenance of the non-critical components of a system must not affect the critical behaviour of the system. In the proposed safety-critical system structure, the mission subsystem cannot (directly) influence the behaviour of the safety subsystem. This restriction on the influence of the mission subsystem introduces the possibility of restricting the effects of any modification to the mission subsystem. In particular, if the structure is rigorously adhered to any modifications to the mission subsystem cannot directly influence the safety subsystem. However, there is the possibility of the mission subsystem indirectly affecting the safety subsystem. For example, if the safety subsystem of a chemical plant assumes that the maximum change in temperature per unit time is bounded by Δtemp , but the mission controller is modified by adding a heating element which can increase the temperature at a greater rate than the assumptions under which the safety subsystem is specified will be violated. However, it is possible to check for this by ensuring that any modification in the mission subsystem does not lead to violations in any assumption over the behaviour of the physical process that is made during production of the specifications of the safety subsystem. Such checks should be possible if all such assumptions are formally documented.

Redundancy

The proposed safety-critical system structure clearly identifies the critical components of the process control system (hence critical computing systems of the process control system). This distinction would allow designers to make a more effective use of redundancy by concentrating on the critical components.

Simpler Interfaces

The proposed safety-critical system structure clearly separates the safety and mission interfaces. This separation allows the specification of simpler critical interfaces. One of the main benefits of a simple safety console interface is that simpler definitions of the

circumstances under which manual intervention is required can be made and suitable warning indicators specified.

2.3.1.3. Requirements Phases

The requirements analysis can be partitioned into five main phases. The phases and the relationship between them is given in figure 2.5. The phases reflect the decision to perform the analysis at two levels (real world and controller) and to partition the analysis into the mission analysis and safety analysis. The phases are discussed in some detail in the following paragraphs.

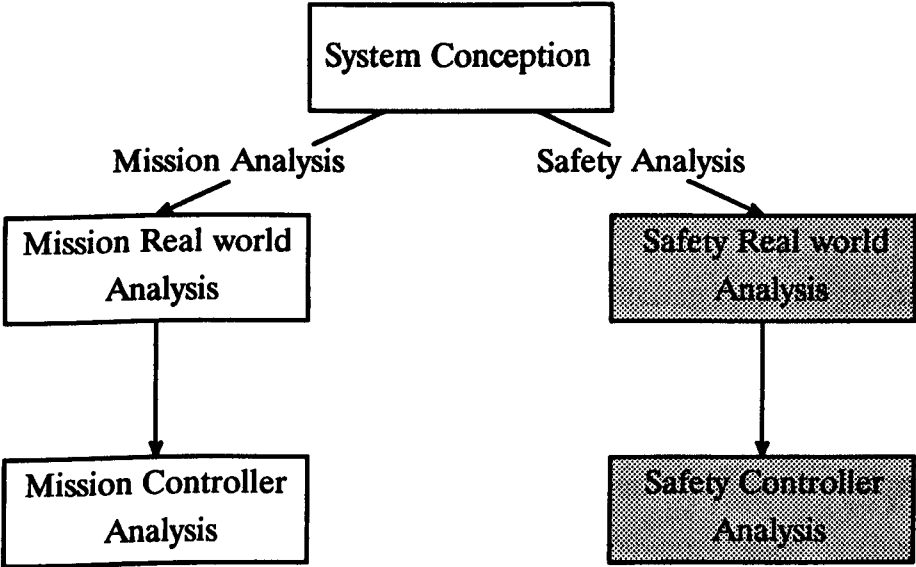


Figure 2.5. Requirement Analysis Phases

System Conception

The starting point for the construction of a safety-critical system (or any system), is the perception of a need for the services that will be provided by the system. This perception is (in the first instance) an informal model of the desired behaviour. This model is usually presented as an informal specification, expressed in a suitable notation. I shall briefly describe the role of informal requirements in process control systems. The (ideal)

informal requirements for a process control system should precisely specify the properties or constraints that the behaviour of a system must satisfy to be acceptable, but not how the behaviour will be realised [Somm82]. For most real systems, it is unrealistic to expect that the informal requirements will satisfy the condition above – or even remain constant during the development of the system. Nevertheless, since the informal requirements will be used as the basis of the analysis (and synthesis) it is important that they be as complete and precise as possible. The system concept phase should include some initial analysis which determines the scope/boundaries of the system. This initial analysis should be performed using a structured analysis approach, such as CORE [Mull79]. For safety-critical systems, in addition, it is desirable for the informal requirements to specify the potential disasters that the system must avoid. As a minimum the informal requirements must specify the safety regulations that the system must adhere to.

Safety Real World Analysis

At the real world level, the safety analysis is performed in three stages which lead to the production of four essential specifications: a *disaster set*, *hazard specification*, *safety real world description* and *safety real world specification*. In the first stage, the potential disasters of the system are identified and specified as a disaster set. The identification of the disasters should be performed using system safety techniques, such as check-lists and Preliminary Hazard Analysis [Rola83]. It is the identification of the disaster set which allows a distinction between the safety and mission related issues to be made. It is essential that the identification of the disaster set be as complete as possible, for *at best* disaster avoidance precautions can be specified for only those disasters that are identified. The second stage also deals with the identification of the possible hazards of the potential disasters of the system. The hazards of the system should be identified by a systematic analysis of the disasters and the possible system conditions. These identified hazards are captured as the hazard specification of the system. The second stage also involves the identification of all the real-world properties and laws which impinge on the safety-critical behaviour of the system and leads to the production of the *safety real world description*.

These properties and laws, and the assumption that the hazards specify all the conditions under which a disaster can occur, are the safety (real world) assumptions of the system. The third stage deals with producing a sufficient condition (the *safety real world specification*) over the behaviour of the system, which if satisfied ensures that none of the disasters can occur under the safety assumptions.

Mission Real World Analysis

At the real world level the mission analysis is performed in two stages which leads to the production of two essential specifications: *mission real world specification* and *mission real world description*. The first stage deals with the identification of the mission orientated behaviour of the system, based on the behaviour specified by the system concept (informal requirements). The mission oriented behaviour of a system has been identified it should be formally specified as the *mission real world specification* of the system. The second stage deals with the identification of all the real-world properties and laws which impinge on the mission-orientated behaviour of the system, and leads to the production of the *mission real world description*. This specification will express the real world assumptions for the mission of the system.

Safety Controller Analysis

The analysis of the safety controller behaviour is performed in two stages which leads to the production of two essential specifications: *safety environment description* and *safety controller specification*. The first stage deals with the identification of the relationships between the sensors and actuators of the safety controller and the properties of the physical process, and leads to the production of the *safety environment description*. This specification will express the safety assumptions of the system. The second stage deals with the identification of the constraints that must be imposed on the behaviour of the sensors and actuators, of the safety controller, for the behaviour of the physical process to comply with the safety real world specification – under the mission assumptions. These constraints on the behaviour of the safety controller are expressed as the *safety controller specification*.

Mission Controller Analysis

At the controller level the mission analysis is performed in two stages which lead to the production of two essential specifications: *mission environment description* and *mission controller specification*. The first stage deals with the identification of the relationships between the sensors and actuators of the mission controller, safety controller and the properties of the physical process, and leads to the production of the *mission environment description*. This specification will express the assumptions for the mission of the system. The second stage deals with the identification of the constraints that must be imposed over the behaviour of the sensors and actuators, of the mission controller, for the behaviour of the physical process to comply with the mission real world specification – under the mission assumptions. These constraints over the behaviour of the mission controller are expressed as the *mission controller specification*.

The above model of the requirement analysis phases has been considerably simplified. For the elicitation of the requirements of most practical systems there will be considerable backtracking between the phases, and in the different stages of a phase. These issues are tackled with the presentation of the development methodology.

2.3.2. Formal Model

In this section it will be argued that to gain a complete understanding of a safety-critical computing system, the requirements of the overall system and the properties of the environment must be analysed in a common formal framework. This section is presented in three parts: firstly, general benefits of using formal methods are discussed; secondly, the benefits of adopting a unified formal framework for all the specifications are stressed; and finally, the essential attributes that a suitable framework must possess are presented.

2.3.2.1. Advantages of Formal Models

Some of the advantages of using formal models during the requirements analysis are discussed in the following paragraphs.

Unambiguity

The importance of developing specifications that are unambiguous (and precise) has been stressed by many researchers [Barn89] and organisations [EWIC85]. In safety-critical systems unambiguous specifications are vital, for many different parties are involved in the development of safety-critical systems. The different parties will have different perspectives on the behaviour of the system; if the specifications are ambiguous these different perspectives may lead to different interpretations of intended system behaviour. By having a formal specification with simple straightforward formal semantics, the possibility of a mistake being caused during the requirements analysis, by misinterpretation is reduced. For example, a precise description of the required behaviour is necessary for the development team to have confidence in the testing criteria derived from the requirements.

Consistency and Completeness

Two problems which are of concern to analysts and designers of systems are those of consistency and completeness [Wass79, Quir85, EWIC85]. A mathematical model will allow the analysts and designers to check formally that the specifications are consistent, in the sense that the different requirements imposed on the system behaviour do not lead to conflicts. Though a mathematical representation can be used to check some aspects of completeness [Jaff89] it is very difficult to show that a requirements specification has specified everything. If we use the analogy between the requirements specification of a system and a painting of a scene; we can infer that a painting is incomplete if it has an incomplete object. However, if an object is missing from the painting we cannot infer this from the painting. Similarly we can infer that a requirements specification is incomplete if it has an incomplete requirement, however if a requirement is missing this cannot be inferred. For safety, the notions of disasters and hazards can be utilized in checks of completeness – by confirming that all the potential disasters of a system have been identified and then confirming that for each disaster all the hazards have been identified.

Formal Verification

By using formal models to represent the specifications the possibility of formal verification [Davi79] is introduced. However, formal verifications of complex systems tend to be very difficult, and in some cases the formal verification process is as complicated as the analysis and development process. This complexity places a question mark over the usefulness of the formal verification of entire systems in system development. A more practical approach may be to concentrate on the formal verification of specific properties of the system, for example the safety of the system.

Automated Development Aids

The use of a formal model in the requirements analysis introduces the potential of automated development aids. However, at present there are no development tools in which there is sufficient confidence for them to be used in safety-critical systems [Mose90]. In the analysis of formal specifications, development tools could be used for: automated checks of consistency and completeness, verification between the specifications at the different phases, simulation of the requirements specification, and the construction of rapid prototypes [Roan86].

2.3.2.2. Disadvantages of Formal Models

Some of the disadvantages of using formal models during the requirements analysis are discussed in the following paragraphs.

Obscure Notation

Formal notations are usually obscure, this arises from the fact formal specification languages are usually experimental. However, if they are to be used for safety-critical systems more attention has to be paid to the readability of specifications. Obscure notations can lead to two problems, firstly, the formalism may create typographic errors, and more importantly they can cause problems in communication between the analysts and the customer.

Expensive

Formal models can be expensive to use. The cost accumulates from having to teach staff how to use these models, this arises from a shortage of skilled people; and the increased effort that must be employed during the requirements analysis, this arises because of the lack of suitable tools.

Lack of Guidelines

Many formal models do not provide methods or guidelines for their use, basically they are formal notations, however formal notations do not solve the problems of constructing specifications. Many problems arise from the inability of analysts to extract information and handle information. Therefore to realize the potential benefits of using formal methods, a systematic development methodology is required – to allow analysts to focus on the realities of the system during requirements analysis and provide a link between the constructs of the formal model and the system properties.

2.3.2.3. Common Formal Framework

In this section I will argue the importance of modelling the environment and requirements of safety-critical systems within a common formal framework.

Communication

A common formal framework for the requirements and environment description reduces the possibility of any errors in communication between the specialist in the application areas and system analysts. The necessity for good communication, between application specialists and system designers, is due to the fact that there is unlikely to be a single person (or single discipline group) with sufficient knowledge of the application area and system design issues to create an overall picture of the system and its environment. The problems caused by poor communication during the elicitation of the requirements have been a major cause in a number of disasters [Leve86].

Test Environment

For safety-critical systems it is clearly infeasible for the developed system to be tested in the physical environment. To test such systems, a simulated test environment is usually constructed; the formal description of the environment can be used to specify the characteristics of the test environment [Chan85].

Assumptions

By constructing a formal description of the environment, assumptions which are made concerning the behaviour of the environment in which a system will work can be stated in a precise notation. This allows interested parties to question these assumptions, before they are used to determine the behaviour required from a system. The problems caused by misunderstandings about assumptions of environmental behaviour have led to several disasters [Leve86]. By having the environment and requirements in the same formal framework the affect of any modifications in the behaviour of the environment (due to a mistake in the original model, or moving the system to a different environment) can be formally analysed.

Critical Properties

By representing the requirements and environment in the same model, the properties of the environment can be used in the construction of the verification proofs for the system. This is particularly important for the formal verification of the safety properties, since it will allow analysers to identify critical assumptions about the environment. Once the critical assumptions have been identified, they can be (formally) documented in the certification.

2.3.2.4. Essential Attributes

In this section I will present four attributes that a formal model must possess, if it is to be used as the basis of the common formal framework.

Physical Laws

The necessity for the model to be able to express physical laws arises from the fact that the behaviour of the environment of the system is expressed using physical laws. To reduce the possibility of mistakes in the representation of the behaviour of the environment, the physical laws must be expressed in a notation which is, as far as possible, the same as the notation used in the application area.

Parallelism

The necessity to explicitly treat parallelism in specifications which include the environment has been argued by many researchers [Gors88]. It is necessary to treat issues related to parallelism in the formal framework during the requirements analysis, since parallelism is present in the environment and system. The parallelism in the environment exists since the real world consists of autonomous entities which exhibit parallel behaviour. Parallelism is included in the requirements specification for two reasons, firstly it allows the designers to use parallelism to meet stringent timing constraints and secondly it simplifies the requirements specification of controllers which will be installed in a parallel environment.

Timing Issues

Timing issues will arise in all of the stages of the requirements analysis. Timing issues are necessary in the description of an environment, since many of the physical laws make an explicit reference to time. Timing issues must be treated during the analysis of real world behaviour of the system when the system must achieve a specified task in a given duration of time. Timing issues must be explicitly handled in the specification of the relationship between the sensors and actuators and the physical properties of the plant. For example, to specify the relationship between the position of a valve and the change in the volume of liquid in a vessel an explicit reference to time is required. Timing issues can arise in the relationship between the sensors and actuators of a controller, if a detailed analysis of the controller is required. For example, consider a valve which must be opened when a thermometer detects that the temperature of the vessel is above a specified value.

For a detailed analysis, the consequences of any delay between the temperature being detected and the valve being opened on the behaviour of the physical process must be determined. The timing constraints which are expressed over the behaviour of the safety actuators and sensors may be critical in the sense that if they are not satisfied the system may enter into a hazardous state.

The pervasive (and critical) nature of timing issues, during the requirements analysis, makes it imperative that the formal model be able to express timing issues in a structured and clear format. It must also be possible to (formally) relate the affect of timing constraints on system behaviour [Dasa85].

Structured Model

A structured model is necessary to handle the specification of complex real-time systems [Hare86]. The structure should be present in the technique used to compose the basic constructs of the model and in the basic constructs. The basic constructs should be represented in a clear concise notation, have clean semantics, and be able to specify the behaviour of the system at a high level. To enhance the readability of the specifications, a graphical representation of the specification should be available.

2.3.3. Development Programme

The development programme must guide the requirements analysis in the context of the framework. In this section a development programme will be introduced for safety-critical systems, in the context of the development model, for which the requirements specifications are represented by the formal model. The necessity for a development programme arises from the fact that there is usually only limited experience in the elicitation of complex requirements in the application areas of safety-critical computing systems, and if any guidelines are available they are usually given in general terms. The major consequences of this are that the elicitation of requirements is performed in an ad hoc way which is difficult to document and monitor; thus confusion can arise between the roles of the different members in the requirements analysis team. The programme will be described in two sections: firstly a development methodology will

be outlined and secondly the role of the members of the requirements analysis team will be discussed.

2.3.3.1. Development Methodology

A formal model to express specifications is only of limited use if no guidelines on how to construct and check the specifications, represented by the model, are given. Furthermore to increase the confidence which can be placed in the accuracy of the requirements specifications, the process of developing the requirements specifications must be systematic and suitable for formal reviews [Redm85]. There are three main principles which underpin the overall requirements analysis scheme, these are:

hierarchical analysis – whenever possible the analysis of the requirements should be performed in a hierarchical way;

early flaw detection – a flaw in the requirements specification should be detected as soon as possible;

verification oriented specifications – the structure of the requirements specifications produced at the end of the different phases, should simplify the formal checks that are performed over the specifications.

Requirements Analysis Processes

Requirements analysis consists of three processes: elicitation, verification and validation [McDe90].

Elicitation

Extraction, usually from the potential users of the system, of information about what the system is required to do. This process is supported by providing prompts in the guidelines which generate questions, for the extraction of information.

Verification

Checking consistency and, as far as possible, completeness of the specification. This process is supported by providing verification conditions, and formal checks for consistency and completeness.

Validation

Checking that the specification accurately reflects the requirements of the potential users, or some authority. This process is supported by providing validation strategies for the real world specifications.

Development methodologies for the real world analysis will be presented in chapter seven, and for the controller analysis in chapter eight.

2.3.3.2. Requirements Analysis Team

An important attribute of safety is that it is a global property of the system, hence system-wide approaches to tackle it are required. Therefore it is essential that any programme to tackle the design of safety-critical systems must be implemented by a multidisciplinary team [Barn89, Leve89]. To ensure that the intentions of the development programme are followed, it is necessary to clearly define the tasks of the members of the requirements analysis team, in relation to the methodologies of the requirements analysis. In the following paragraphs I will discuss a possible development scheme, in terms of the main teams involved during the requirements analysis.

Customer

The customer of a system is the individual, team or organization which has commissioned the system. The task of the customer is to supply a high-level (informal) specification of the mission requirements and the safety requirements of the system, in the form of a system concept. The safety portion of the system concept need only be stated in general terms, e.g., for an aircraft, the customer may demand: “the system must satisfy all the safety requirements imposed by the CAA”. The customer’s view of the formal requirements specification is in two parts: i) the safety-critical specification represented by the safety real world description and safety real world specification; and ii) the mission-oriented specification represented by the mission real world specification and mission real world description.

Disaster Analysis Team

This is a team of application specialists and safety experts. The task of the disaster analysis team of a system is to identify an initial real world description of that system the disasters set of the system by an analysis of the system concept.

Hazard Analysis Team

This team, like the disaster analysis team, consists of application specialists and safety experts. The task of the team is to identify and specify the hazard specification, safety real world description and safety real world specification by an analysis of the initial real world description and disaster set. This team (together with the disaster analysts) must then verify that if the behaviour specified by the safety real world specification is maintained then a disaster will not occur.

Mission Real World Team

This is a team of system analysts and application specialists. The task of the mission analysis team is to construct the mission real world description and mission real world specification by an analysis of the system concept and initial real world description. The team (together with application specialists and customer) must validate the mission real world specification. If the customer wishes to confirm that the mission is safe, in terms of the safety real world specification, the team must verify that a behaviour that satisfies the mission real world description and mission real world specification satisfies the safety constraint.

Safety Controller Team

This is a team of analysts experienced with the safety controller methodology (see later). The task of the team is to construct and verify the safety environment description and safety controller specification by an analysis of the safety real world description and safety controller specification.

Mission Controller Team

This is a team experienced with the mission controller methodology (see later). The task of the team is to construct and verify the mission environment description and mission controller specification by an analysis of the mission real world description and mission controller specification.

Safety Operator

The safety operator does not play a direct role in the elicitation of the requirements, but may suggest changes to the mission controller specification, or safety controller specification. Any changes to the mission controller specification suggested by the safety operator must be confirmed by the customer before being accepted. Any changes to the safety controller requirements would have to be checked by the hazard analysis team before being accepted. The benefits of giving the operator a formal opportunity to influence the safety controller specification are: i) that the experience of the safety operator can be useful in the specification of the operator interface and the specification of schemes for manual intervention; and ii) yet any of his suggestions can be checked.

Mission Operator

The mission operator team does not play a direct role in the development methodology, but may suggest changes to the mission controller requirements. Any changes suggested by the mission operators must be confirmed, by the customer, before being accepted.

Certification Body

The certification body is the regulating authority of the application area into which the computer system will be installed. If the regulating authority has a graded certification scheme then the authority must specify the safety grade for the system. Once the requirements analysis of the system has been completed by the development team, the authority must certify the specifications. The certification is performed in two stages: i) the authority must certify that the behaviour defined by the safety real world specification

precludes the possibility of a disaster, ii) the authority must certify that satisfaction of the safety controller specification (under the safety environment description) implies satisfaction of the safety real world specification.

Management

The management (team) are a team of individuals who are trained in the co-ordination and management of the development methodology. The task of management is to ensure that the development team adheres to the guidelines given in the development methodology. In particular, to ensure that each phase is completed satisfactorily before proceeding to the next stage.

Table 2.1: Organisational Roles in Requirements Analysis

Team	Input Specifications	Output Specifications
Customer	-	System concept
Disaster Analysts	System Concept	Initial real world description Disaster set
Hazard Analysts	System Concept Initial real world description Disaster Set	Hazard specification Safety real world description Safety real world specification
Mission real world	System concept Initial real world description	Mission real world description Mission real world specification
Safety controller	Safety real world description Safety real world specification	Safety environment description Safety controller specification
Mission controller	Mission real world specification Mission real world description	Mission environment description Mission controller specification

2.4. Working Example

To clarify the main concepts of the specification model, as presented in the following chapters, the specifications produced during the requirements analysis of a simple chemical plant “the reaction vessel” will be discussed. The system concept for this example control system is given below.

System Concept

“A computer system which can control the reaction vessel, illustrated in figure 2.6, is required. Once the computer system has been installed, the reaction vessel must be able

to react a specified volume of chemical A with a specified volume of chemical B, producing a chemical C. For the reaction between A and B to take place the temperature of the vessel must be raised above a specified activation value. To achieve a satisfactory yield the temperature of the vessel must be kept close to the activation temperature. The operator must be able to specify the reaction volumes (of A and B), and since the activation temperature depends on the relative volumes he must also be able to adjust the activation temperature. During the lifetime of the system a light indicating the status of the reaction vessel to the operator is required. The relationship between the colour of the light and the status of the reaction vessel should obey the following (informal) rules:

- a) it should be *green* when the vessel does not contain the required volumes of A and B.
- b) it should be *amber* when the vessel does contain the required volume of A and B and the reaction has not started and
- c) it should be *red* while the reaction is in progress.

The mission (or safety) operator interacts with the mission controller via the plant dial – to specify the sequences in which tasks are performed in the reaction vessel.

Furthermore it is known that the chemicals A and B are hazardous, hence the chemical plant must be certified by a licensing authority. The computing system must ensure that the overall chemical plant will not enter into a hazardous state. The safety operator interacts with safety controller via the safety dial. During the lifetime of the system a light indicating the safety status of the reaction vessel is required. ” The vessel will be referred to as RV for brevity.

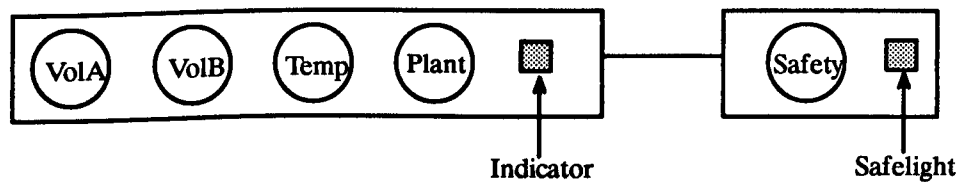
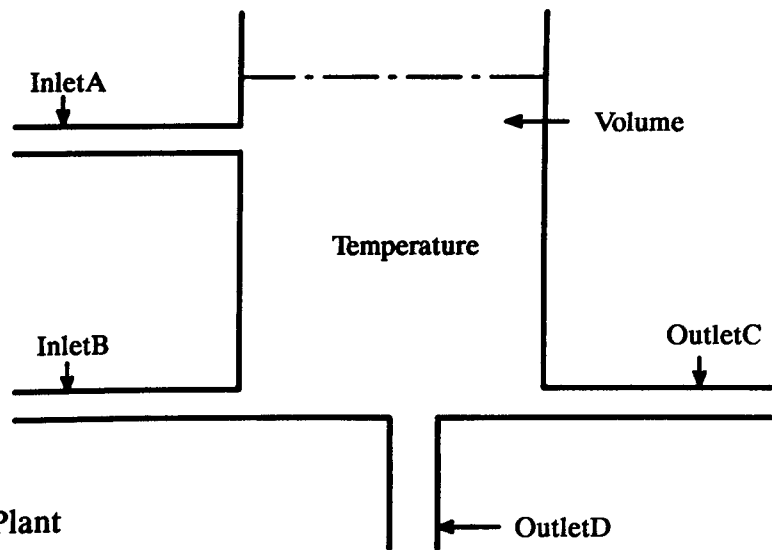


Figure 2.6. Reaction Vessel

2.5. Summary

This chapter has discussed the role of requirements analysis in the development of safety-critical computing systems. The problems which arise during requirements analysis were outlined, and some of the current approaches were briefly discussed. The role of the requirements specification was discussed in relation to the subsequent development stages. Two view points were identified: the customer's and the developer's. These were related to system behaviour at two levels; the customer's viewpoint was defined in terms of *real-world* properties and the developer's viewpoint in terms of *controller* properties.

A structured approach to requirements analysis was outlined. The approach consisted of three main parts: a development model, a formal model and a development programme. The development model outlined a general structure for safety-critical computing systems

which represents a clear distinction between the safety-critical and mission-oriented components of the system. The development model also identified the main stages during requirements analysis and essential specifications produced by each stage. The main attributes that a suitable formal model must possess were discussed these included: the ability to represent the behaviour of the environment, the ability to represent timing requirements and some means to allow specifications to be represented in a coherent format. The development programme discussed the role of the methodologies which guide the development of the requirements specification within the development model; and provided an outline of the organisational roles with respect to the main stages of the requirements analysis.

Chapter 3 - Basic Concepts

In this chapter, I will discuss the basic concepts of the proposed formal specification model. An abundance of formal models have been proposed for real-time systems, therefore a brief justification for proposing another model in this thesis, is appropriate at this point. The aim of this thesis is to investigate the role of formal methods in requirements analysis as opposed to determining which model is most suitable. Hence the view was taken that by a consideration of the essential specifications; and the class of systems under consideration an appropriate formal model to study the role of formal methods should be constructed. Such a model should be a minimal model for the requirements analysis of the class of systems under consideration. Firstly, I will discuss how the time base and state space of a model of a system are defined, and give the definition of the universal history set of a system in terms of the time base and state space. Secondly, I will discuss the formal representation of a history description. Finally, a set of satisfaction conditions which can be used to impose constraints over a set of histories, will be introduced.

3.1. Time

One of the first decisions that must be made in the construction of a specification model for real-time systems, in which timing constraints will be expressed, concerns the representation of time in the model. Different time structures for the basis of tense (temporal) logic have been studied in philosophy, linguistics [Bent80, Bugr84, Kuhn89] and computing science [Bern81, Miln83, Jaha86, Reed86, Pnue88, Jose89, Ostr90]. A basic issue of discussion is whether a discrete or dense representation of time should be used. In the following paragraphs I will discuss some of the benefits of dense and discrete representations of time; after which I will briefly justify the time base to be used for the proposed formal model.

Dense Time

Several formalisms have chosen a dense time base, these include extensions to qualitative Temporal Logic (TL) [Bern81, Pnue88], extensions to Communicating Sequential Processes (CSP) [Reed86], extensions to Calculus of Communicating Systems (CCS) [Miln83] and general models of computation [Jose89].

Four attributes of dense time which make it attractive for a time base of a general specification model for process control systems are discussed below.

i) *Physical laws*. The model should be able to express physical laws which govern the properties of the environment, that impinge on the behaviour of the system. Many of these physical laws are defined in terms of physical time, e.g., the distance travelled by a train for a duration of time is given by an integral over the velocity of the train.

ii) *Performance measures*. In the performance analysis of process control systems time has been traditionally been treated as a continuous variable. The following quote [Ho89] gives some justification the treatment of time as a continuous variable in a general specification model for process control systems: “Performance measures related to Discrete Event Dynamic Systems (DEDS) are usually formulated in terms of continuous variables, such as average throughput, waiting time. In fact “time” (a fundamental performance measure) is by definition and convention a continuous variable. There is little technological or mathematical advantage in considering a discrete time or “sample–data” model of DEDS. In fact, many tools, such as queuing network theory or perturbation analysis, explicitly rely on the smoothing properties of the “expectation” or “average” operator to make analysis possible.”

iii) *Minimum separation*. The real world of a process control system consists of many concurrent interacting processes, for such a system it is not possible to give a minimum separation between events in the different processes. The lack of minimum separation makes it difficult to define a *granularity of time* for a discrete time base, hence a dense representation of time is more suitable for the real world level.

iv) *Expressiveness*. In the context of this work a formalism with a dense time base, can express the properties of a similar formalism with a discrete time base.

Discrete Time

Several formalisms have chosen a discrete time base, these include extensions to TL [Pnue88], CCS [Miln83], fair transitions systems [Ostr90], first order predicate calculus [Jaha86] and Petri-nets [Merl76, Ramc74].

Two attributes of discrete time which make it an attractive time base for a general formal model for process control systems are discussed below.

i) *Discrete Systems*. Computers are discrete systems, for which the internal actions occur in discrete time. Furthermore, since the observations and control commands of the system can only occur in synchrony with the executions of commands by the control system, they will lie in the same domain as the other actions of the system [Jose90]. Hence it can be argued that a discrete time base is a convenient time base when we are concerned with the behaviour at the controller level.

ii) *Simpler Models*. A discrete representation of time can lead to simpler models. For example, Real-Time Logic (RTL) which is described by Jahanian and Mok [Jaha86] has a discrete time base, this allows to transform formulas of RTL into a restricted Pressburger arithmetic for which decision procedures are available.

Base Time

If a discrete representation of time is used the notion of a granularity of time would have to be introduced. However events at the real world level can be arbitrary close. Although by choosing a small enough time-grain the analysis of arbitrary close events may be performed in discrete time, as has been suggested by Milner [Miln83], analysis at the “real world” level would be complicated by having to treat the continuous laws of physics in a discrete time framework. If a dense time base is used the behaviour at the real world and controller level can be represented, in a single time framework. The base time set (BT) for the proposed specification model will be the set of non-negative real numbers. It should be noted that it is not being suggested that a dense representation of time is the most suitable

representation of time for all the development stages of any real-time system. Rather it is being suggested that a dense representation of time (in particular the set of non-negative reals) is a suitable representation for the requirements analysis of the class of safety-critical systems considered in the thesis.

3.1.1. Time Points

A time point is an element of BT, the base time set and will be denoted by t ; if more than one time point is required in an expression then the time points will be subscripted as: t_0, t_1, \dots . From the properties of real numbers we can infer the following four properties of time points in the base time.

- i) The binary relation of *temporal precedence* ($<$, less than) over the time set is *transitive* (i.e., $\forall t_0, t_1, t_2 \in BT: (t_0 < t_1 < t_2 \Rightarrow t_0 < t_2)$) and *irreflexive* (i.e., $\forall t \in BT: \neg(t < t)$).
- ii) The binary relation of *weak temporal precedence* (\leq , less than or equal to) over the time set is *transitive* (i.e., $\forall t_0, t_1, t_2 \in BT: (t_0 \leq t_1 \leq t_2 \Rightarrow t_0 \leq t_2)$), *reflexive* (i.e., $\forall t \in BT: t \leq t$) and *antisymmetric* (i.e., $\forall t_0, t_1 \in BT: (t_0 \leq t_1 \wedge t_1 \leq t_0 \Rightarrow t_0 = t_1)$).
- iii) The property of *linearity* (i.e., $\forall t_0, t_1 \in BT: (t_0 < t_1 \vee t_1 < t_0 \vee t_1 = t_0)$) holds for the time set.
- iv) The property of *denseness* (i.e., $\forall t_0, t_1 \in BT: t_0 < t_1 \Rightarrow \exists t_2 \in BT: t_0 < t_2 < t_1$) holds for the time set.

3.1.2. Time Intervals

Within BT various kinds of intervals may be defined, all the intervals will be subsets of BT. An interval will be denoted by Int; and if more than one interval is required in an expression then the intervals will be subscripted as: Int_0, Int_1, \dots .

Definition: Time intervals

A subset of BT is a time interval if and only if for any two time points in the subset all time points from the set BT which lie between the two time points are elements of the subset.

More precisely, $Int \subseteq BT$ is a *time interval* iff $\forall t_0, t_1 \in Int: \forall t_2 \in BT: (t_0 < t_2 < t_1 \Rightarrow t_2 \in Int)$.

The following corollary follows from the definition of time intervals.

Corollary 3.1

A time interval of BT satisfies the temporal precedence, weak temporal precedence, linearity and denseness conditions. (Clearly BT is itself a time interval).

Proof

The *temporal precedence*, *weak temporal precedence* and *linearity* conditions hold for a time interval, since a time interval is a subset of BT. The *denseness* property holds for a time interval since for any two time points in the set all time points which lie between the two time points are elements of the interval.

With two time points t_0 and t_1 , four classes of intervals of BT can be defined:

Open intervals: $(t_0, t_1) =^{\text{def}} \{t \in BT \mid t_0 < t < t_1\}$, $t_0, t_1 \in BT$, $t_0 < t_1$;

Closed intervals: $[t_0, t_1] =^{\text{def}} \{t \in BT \mid t_0 \leq t \leq t_1\}$, $t_0, t_1 \in BT$, $t_0 \leq t_1$;

Half-open intervals: $(t_0, t_1] =^{\text{def}} \{t \in BT \mid t_0 < t \leq t_1\}$, $t_0, t_1 \in BT$, $t_0 < t_1$ and

Half-closed intervals: $[t_0, t_1) =^{\text{def}} \{t \in BT \mid t_0 \leq t < t_1\}$, $t_0, t_1 \in BT$, $t_0 < t_1$.

For an interval Int, we define a set $SI(Int)$ which is the set of all closed time intervals, included within Int. From the definition of a time interval we can infer the following two properties of the set $SI(BT)$,

i) The *temporal precedence relation* can be extended to cover the *intervals* in this set as follows: $\forall Int_0, Int_1 \in SI(BT): Int_0 < Int_1$ iff $\forall t_0 \in Int_0, \forall t_1 \in Int_1: t_0 < t_1$.

ii) Similarly we can define the notion of *weak (interval) temporal precedence* as:

$\forall Int_0, Int_1 \in SI(BT): Int_0 \leq Int_1$ iff $\forall t_0 \in Int_0, \forall t_1 \in Int_1: t_0 \leq t_1$.

Unless otherwise stated interval should be understood as a closed one. It should be clear, that the temporal precedence and weak temporal precedence conditions hold for $SI(BT)$. However, the linearity condition does not hold for $SI(BT)$.

Boundary Points

For any interval, two boundary points can be defined: the *start* and *end* points. Roughly speaking, the start point is the first time point of the interval and the end point is the last time point of the interval. The formal definitions are given below.

Definition: Start point

The start point of an interval Int (denoted by $s(Int)$) is the earliest time point in the interval.

More precisely, $s(Int) = t$ s.t. $\forall t_0 \in Int: t \leq t_0$.

Definition: End point

The end point of an interval Int (denoted by $e(Int)$) is the latest time point in the interval.

More precisely, $e(Int) = t$ s.t. $\forall t_0 \in Int: t \geq t_0$.

Duration

We will define a measure of the length of any interval called the duration of the interval. Roughly speaking, the duration of an interval is the length of the period of time represented by the interval. The formal definition is given below.

Definition: Duration

The duration of an interval Int (denoted by $dur(Int)$) is the difference between the end point and start point of the interval.

More precisely, $dur(Int) = e(Int) - s(Int)$.

Intersection and Unions

The intersection (and union) of two time intervals will only be an interval if they overlap as expressed in the following lemmas.

Lemma 3.1

The set given by the intersection of two intervals (say Int_0, Int_1) is a interval if and only if the intersection is non-empty.

More precisely, $\forall Int_0, Int_1 \in SI(BT): (Int_0 \cap Int_1) \in SI(BT)$ iff $(Int_0 \cap Int_1 \neq \emptyset)$.

Proof. Immediate.

Lemma 3.2

The union of two intervals (Int_0, Int_1) is a interval if and only if their intersection is non-empty.

More precisely, $\forall Int_0, Int_1 \in SI(BT): (Int_0 \cup Int_1) \in SI(BT)$ iff $(Int_0 \cap Int_1 \neq \emptyset)$.

Proof. Immediate.

3.1.3. System Lifetime

The system (operational) lifetime is the time interval of BT given by $T(SY)$, where $s(T(SY))$ represents the time point at which the system starts operation and $e(T(SY))$ is the time point at which the system is finally closed down. In other words, the system lifetime is an interval of the time set, which represents the operational lifetime of the system. If the system is obvious from the context the system lifetime will be denoted by T . For example, the system lifetime of the reaction vessel is given by the T , where $s(T)$ is the time point at which the reaction vessel starts operation and $e(T)$ is the time point at which the reaction vessel is finally closed down.

3.2. State Variables

The state of a system will be given by the values of its state variables. These are the (time varying) quantities which either have a significant effect on, or measure factors which affect, the mission or the safety of a plant, e.g., pressures, temperatures and valve settings. The vector of all the state variables for a system will be referred to as the state vector and referred to as S_v . A state vector with n state variables, for the system SY , is denoted by: $SY.S_v = \langle p_1, ..., p_n \rangle$, $p_i \neq p_j$, for $i \neq j$, where p_i, p_j represent state variables $i, j \in \{1, ..., n\}$. The set of the state variables for a system will be referred to as the state set. The number of state variables in the state vector is referred to as the state number and is denoted by $SY.n$. A state set with n variables, for the system S , is denoted by $SY.S = \{p_1, ..., p_n\}$. For a given system (SY) $SY.S_v$ and $SY.S$ are the same except the former is an ordered sequence and the latter is a set. Both the state vector and state set represent the variables (i.e., names, identifiers etc), and not the actual values (these will be defined later).

State Variables Table

The state variables of a system can be concisely represented in a tabular format – as the state variables table. For each state variable of the system there must be a row in the state variables table. The table has four columns; the headings and a brief description of the content of each column is given below.

- i) *Variables*: specifies the notation used to represent the variable, of the form p_i .

- ii) *Units*: specifies the units in which the values of the variable are measured.
- iii) *Name*: specifies the name to be used in informal discussions.
- iv) *Comments*: briefly justifies the introduction of the variable.

For example, in the specification of the reaction vessel twenty nine state variables are required, the state vector is denoted by $RV.Sv = \langle p_1, ..., p_{29} \rangle$, and the corresponding state set by $RV.S = \{p_1, ..., p_{29}\}$. A detailed description of the reaction vessel's state variables is given in table 3.1.

Table 3.1: State Variables of Reaction Vessel

Notation	Units	Name	Comments
p1	seconds (s)	Clock	A perfect clock of the system.
p2	dm ³	VolA select	The selector on the operator interface for the volume of A set point.
p3	dm ³	VolB select	The selector on the operator interface for the volume of B set point.
p4	°K	Temp select	The selector on the operator interface for the liquid temperature set point.
p5	Op_setting	Plant select	The selector on the operator interface that allows the operator to control the sequence of operations of the reaction vessel.
p6	°K	Temperature	The temperature of the contents of the vessel.
p7	dm ³ /s	FlowA	The flow rate of liquid A into the vessel.
p8	dm ³ /s	FlowB	The flow rate of liquid B into the vessel.
p9	dm ³ /s	Flow	The flow rate into the vessel.
p10	dm ³	VolumeA	The volume of liquid A in the vessel.
p11	dm ³	VolumeB	The volume of liquid B in the vessel.
p12	dm ³	VolumeC	The volume of liquid C in the vessel.
p13	dm ³	Volume	The volume of liquid in the vessel.
p14	dm ³ /s	OutflowC	The flow rate out of the vessel, via OutletC.
p15	dm ³ /s	OutflowD	The flow rate out of the vessel, via OutletD.
p16	dm ³ /s	Outflow	The flow rate out of the vessel.
p17	Sa_setting	Safety select	The selector on the operator interface that allows the operator to set the VolA set point.
p18	L_state	Indicator	The indicator light on the mission operator interface.
p19	L_state	Safelight	The indicator light on the safety operator interface.
p20	Ex_rating	Explosion	The state of the system in terms of the presence or absence of an explosion.
p21	Lock_set	LockA	The lock for InletA.
p22	Lock_set	LockB	The lock for InletB.

p23	mm	ValveD	The valve which controls the flow rate out of OutletD.
p24	°K	Thermometer	The thermometer which measures the temperature of the contents of the vessel.
p25	mm	ValveA	The valve which controls the flow rate through InletA.
p26	mm	ValveB	The valve which controls the flow rate through InletB.
p27	mm	ValveC	The valve which controls the flow rate through OutletC.
p28	Reg_State	Regswitch	The device which controls the state of the regulator.
p29	Reg_Set	Regtemp	The device which is used to set the working temperature of the regulator.

3.2.1. Variable Ranges

The set of possible values, for a state variable, will usually be determined by physical laws or construction limitations. This set will be referred to as the variable range and for variable p_i will be denoted by V_{p_i} .

For example, consider the following two cases from the specification of the reaction vessel.

i) Suppose that the temperature of the liquid (in the reaction vessel) must be between T_l and T_u °K. Then the range set of the corresponding states variable (p_6) is given as:
 $V_{p_6} = \{x \in R | T_l \leq x \leq T_u\}$. This is an example of a constraint imposed by the physics of the environment.

ii) The state variable (p_{24}) representing the digital thermometer must represent only the values the thermometer can read. Let us suppose the thermometer can read values in the range π to $\pi + \Pi.\Delta\pi$ °K, with a granularity of $\Delta\pi$ °K

The range set is given as: $V_{p_{24}} = \{\pi_l, \pi_l + \Delta\pi, \pi_l + 2.\Delta\pi, ..., \pi + \Pi.\Delta\pi\}$.

This is an example of a constraint imposed by the construction of the plant.

Ranges Table

The ranges of the state variables of a system can be concisely represented in a tabular format – as the ranges table. The ranges table has three columns; the headings and a brief description of the content of each column is given below.

- i) *Variables*: specifies the variables.
- ii) *Range*: specifies the range of the variables.

iii) *Comments*: briefly justifies the range of the variables.

For example, the ranges of the state variables of the reaction vessel are described in detail by table 3.2.

Table 3.2: Variable Ranges of Reaction Vessel

Notation	Range	Comments
p1	T	The range of this clock is the system lifetime.
p2, p3	{0, Δl, 2.Δl, ... , L.Δl}	Where Δl is a number (Δl > 0) that represents the granularity of the volume selectors and L.Δl represents the maximum value.
p4	{k ₁ , ..., k _r }	Where k ₁ , ..., k _r represents the r possible values of the temperature selector.
p5	{off, on, start, collect, end}	The range set represents the five possible states of Plant select.
p6	{x ∈ R T _l ≤ x ≤ T _u }	T _l and T _u are the upper and lower limits of the temperature of the liquid in the vessel.
p7	{x ∈ R 0 ≤ x ≤ FmaxA}	FmaxA is the maximum flow rate of liquid A into the vessel.
p8	{x ∈ R 0 ≤ x ≤ FmaxB}	FmaxB is the maximum flow rate of liquid B into the vessel.
p9	{x ∈ R 0 ≤ x ≤ Fmax}	Fmax is the maximum flow rate into the vessel.
p10, p11, p12, p13	{x ∈ R 0 ≤ x ≤ Vmax}	Vmax is the maximum level of liquid that the vessel can contain.
p14	{x ∈ R 0 ≤ x ≤ OmaxC}	OmaxC is the maximum flow rate out of the vessel, via OutletC.
p15	{x ∈ R 0 ≤ x ≤ OmaxD}	OmaxD is the maximum flow rate out of the vessel, via OutletD.
p16	{x ∈ R 0 ≤ x ≤ Omax}	Omax is the maximum flow rate out of the vessel.
p17	{off, reset, on}	The range represents the three possible states of the safety set switch.
p18, p19	{g, a, r}	Where g: green a: amber and r: red
p20	{false, true}	False represents the absence of an explosion; and true represents the presence of an explosion.
p21, p22	{on, off}	The locks can be in two states on or off.
p23	{x ∈ R 0 ≤ x ≤ Dmax}	Where Dmax is the maximum width to which ValveD can be opened.
p24	{π, π + Δπ, ... , π + Π.Δπ}	Where π is the minimum temperature reading, π + Π.Δπ is the maximum temperature reading and Δπ the granularity.
p25	{x ∈ R 0 ≤ x ≤ Amax}	Where Bmax is the maximum width to which ValveB can be opened.
p26	{x ∈ R 0 ≤ x ≤ Bmax}	Where Bmax is the maximum width to which ValveB can be opened.
p27	{x ∈ R 0 ≤ x ≤ Cmax}	Where Cmax is the maximum width to which ValveC can be opened.

p28	{on, off}	On and off are the two states of the regulator,
p29	{k ₁ , ..., k _r }	As in p ₃ .

State Space

The state space of a system SY is denoted as SY. Γ , and is given by the cross product of the variable range set. That is, for a system with n state variables, the state space (Γ) is given as: $\Gamma = V_{p_1} \times V_{p_2} \times \dots \times V_{p_{n-1}} \times V_{p_n}$. An element of the state space is referred to as a state value (or state) and denoted by V (V_1, V_2, \dots).

3.3. Variable Categories

One of the main differences between the customer’s and developer’s view of the requirements specification is in the state variables over which behaviour is specified. To clarify this distinction, in terms of the states space, three broad variable categories are introduced.

Physical Variables

The physical variables are the state variables which represent the physical properties of the plant. The set of physical variables is denoted by P and the vector by Pv.

Definition: Physical variables

The physical variables of a system are those state variables which represent the state of the physical process.

For example, in the reaction vessel we have: $P_v = \langle p_1, p_6, \dots, p_{16}, p_{20} \rangle$.

Operator Variables

The operator variables are the state variables which represent the selectors and indicators of the operator console. The set is denoted by Op and the vector by Opv.

Definition: Operator variables

The operator variables of a system are those state variables which represent the state of the operator console.

For example, in the reaction vessel we have: $Op_v = \langle p_2, p_3, p_4, p_5, p_{17}, p_{18}, p_{19} \rangle$.

Transducer Variables

The transducer variables are the state variables which represent the sensors and actuators of the controller. The set is denoted by T_d and the vector by T_{dv}

Definition: Transducer variables

The transducer variables of a system are those state variables which represent the state of the sensors and actuators.

For example, in the reaction vessel we have: $T_{dv} = \langle p_{21}, \dots, p_{29} \rangle$.

Real World Variables

The customer view of the requirements specification is given by the specification of constraints over the physical and operator variables. Collectively, the physical and operator variables will be referred to as the real world variables. The real world variable set is denoted by R and the vector by R_v . The real world variables can be partitioned into two categories which reflect the partition of safety and mission issues in the safety-critical system structure. These are discussed below.

Safety Real World Variables

The safety real world variables are the real world variables of a system over which the safety real world specification is imposed. As such the set (or vector) of safety real world variables is defined during the safety real world analysis. The set is denoted by SR and the vector by SR_v .

Definition: Safety real world variables

The safety real world variables of a system are those state variables which represent the critical behaviour of the process or the state of the safety operator console.

For example, in the reaction vessel we have: $SR_v = \langle p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{17}, p_{19}, p_{20} \rangle$.

Mission Real World Variables

The mission real world variables are the real world variables of a system over which the mission real world specification is imposed. As such the set (or sequence) of mission real world variables is defined during the mission real world analysis. The set is denoted by MR

and the vector by MRv.

Definition: Mission real world variables

The mission real world variables of a system are those state variables which represent the mission-oriented state of the process or state of the mission operator console.

For example, in the reaction vessel we have: $MRv = \langle p_1, \dots, p_{16}, p_{18} \rangle$.

Controller Variables

The developer's view of the requirements specification is given by the specification of constraints over the transducer and operator variables. Collectively the transducer and operator variables will be referred to as the controller variables. The set is denoted by C and the vector by Cv. The controller variables can be partitioned into two categories which reflect the partition of safety and mission issues in the safety-critical system structure.

Safety Controller Variables

The safety controller variables are the controller variables of a system over which the safety controller specification is imposed. As such the set (or sequence) of safety controller variables is defined during the safety controller analysis. The set is denoted by SC and the vector by SCv.

Definition: Safety controller variables

The safety controller variables of a system are those state variables which represent the state of the safety console or safety transducers.

For example, in the reaction vessel: $SCv = \langle p_{17}, p_{19}, p_{21}, p_{22}, p_{23}, p_{24} \rangle$.

Mission Controller Variables

The mission controller variables are the controller variables of a system over which the mission controller specification is imposed. As such the set (or sequence) of mission controller variables is defined during the mission controller analysis. The set is denoted by the set MC and the vector by MCv.

Definition: Mission controller variables

The mission controller variables of a system are those state variables which represent the state

of the mission console or mission transducers.

For example, in the reaction vessel: $MC_v = \langle p_2, p_3, p_4, p_5, p_{18}, p_{25}, p_{26}, p_{27}, p_{28}, p_{29} \rangle$.

Variable Relationships

In this section I will present the relationships between the subsets of the state variables. The physical, operator and transducer variable subsets are:

- i) complete with respect to the system state variables set (i.e., $P \cup Op \cup Td = S$);
- ii) mutually disjoint (i.e., $P \cap Op = \emptyset$, $P \cap Td = \emptyset$ and $Td \cap Op = \emptyset$).

The subsets of the real world set ($R = P \cup Op$) are complete with respect to the real (i.e., $SR \cup MR = R$). The subsets of the controller set ($C = Op \cup Td$) are complete with respect to the controller set and are disjoint (i.e., $SC \cup MC = C$ and $SC \cap MC = \emptyset$).

3.4. System History

Given the lifetime T and the state space Γ of a system, a model of the behaviour of a system during an interval of time can be constructed as an evolution. An *evolution* through the states of a system during an interval is a mapping from each time point in the interval to a system state. For example, the evolution Ev during the interval $[t_0, t_1]$ is a mapping of the form $Ev: [t_0, t_1] \rightarrow \Gamma$ (the mapping is illustrated in figure. 3.1).

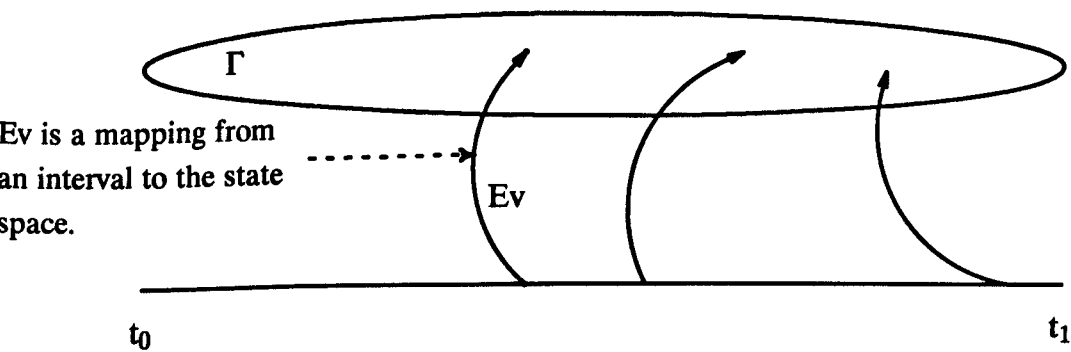


Figure 3.1. An Evolution

An evolution of the system which starts at the start point of the system ($s(T)$) and ends at the end point of a system ($e(T)$) is referred to as a *history* of the system. Most process control systems are unpredictable, in the sense that the history of the system is not known in advance. Therefore when reasoning about such systems all “possible” histories must be

considered. This set of histories is called the universal history set of a system, and specifies all possible behaviours of the system.

3.4.1. History Function

For every possible history (H) of the system, the sequence of values taken by a state variable p_i can be represented as a function $H.p_i: T \rightarrow V_{p_i}$. Hence a history itself is the mapping $H: T \rightarrow \Gamma$, such that $H(t) = \langle H.p_1(t), \dots, H.p_n(t) \rangle$. With every history H there is an associated set (denoted by H as well): $H = \{ \langle s, t \rangle : t \in T \wedge s = H(t) \}$. The set H consists of a set of *snapshots*. A snapshot is a tuple consisting of an evaluated state vector and the time of the evaluation, for a given history. For example, the snapshot at time t , for history H , is given by the vector $\langle H.p_1(t), \dots, H.p_n(t), t \rangle$. The snapshot space is denoted as $\Gamma' (= \Gamma \times T)$. The universal history set of a system is denoted by ΓH and is the set of all functions $H: T \rightarrow \Gamma$. Arbitrary subsets of the universal history set will be referred to as history sets and denoted by HH .

To simplify references to the histories it will be assumed that a labelling function ($LfH(H)$) which maps a history to a label can be defined. More precisely the function is such that $LfH: \Gamma H \rightarrow \{\text{labels}\}$ and $LfH(H) \neq LfH(H')$ for all $H \neq H'$ (i.e., LfH is an injective function). The histories can then be referenced by using the labels as indices. The indexed histories ($H_x, x \in \{\text{labels}\}$) can be defined using the labelling function, where $H_{LfH(H)} = H$, for all $H \in \Gamma H$.

Restricted History Functions

In some cases, for a particular history, we may only be interested in the snapshots during a specified interval of the system lifetime (i.e., a particular evolution). Or alternatively, we may only be interested in the value of a particular subvector during the system lifetime.

A *restricted domain* version of the history function, for a specified interval, can be stated using the (standard) restriction operator. For example, the restricted version of a

history H , for the interval Int , is denoted by $H|_{Int}$, where $H|_{Int}: Int \rightarrow \Gamma$, and $\forall t \in Int: H|_{Int}(t) = H(t)$.

A *restricted range* version of the history function which returns only the values of a specified variable sequence will be defined using a “.” notation, where the subvector of interest is specified after the “.”. For example if we are only interested in the vector $pv (= \langle p_1, p_2, p_3 \rangle)$ the restricted history function would be stated as $H.pv$:
where $H.pv: T \rightarrow V_{p_1} \times V_{p_2} \times V_{p_3}$, and $\forall t: H.pv(t) = \langle H.p_1(t), H.p_2(t), H.p_3(t) \rangle$.

The restricted range and restricted domain functions may, of course, be combined. For example, the condition *that for history H during the interval Int the variables in vector pv must be equal to $\langle x, y, z \rangle$* may be formally stated as $H.pv|_{Int} = \langle x, y, z \rangle$. (A neater construct to express such conditions is defined later.)

3.5. History Descriptions

In this section I will introduce the formal construct which will be used to define the environment descriptions, in terms of the universal history set. The construct will be explained by discussing how it can be used to specify restrictions imposed on system behaviour, by the physical laws which govern the physical process; and the relationship between the sensors, actuators and physical process which follow from the construction of the controller.

Recall that the universal history set of a system specifies the set of all possible histories of the system, given two fundamental constraints over the behaviour i) the system lifetime and ii) the variable range constraints. However, if we consider the behaviour of the system in the context of its environment then many of the histories in the universal history set could not occur due to the laws of physics and the construction of the machinery. Some of these restrictions can be formally expressed using the variable ranges, but to formally express most of the restrictions, more flexible (in terms of expressive power) constraints (relations) are required. In the model three description relations are provided: i) *variable class relations*, ii) *invariant relations* and iii) *history relations*.

The formal semantics of the relations will be so formed, that for every relation we are able to say whether a particular function $H: T \rightarrow \Gamma$ satisfies it or not. These relations will be used in the construction of a history description.

Definition: History Description

A history description Desc is a six-tuple $Desc = \langle T, Sv, VP, CP, IR, HR \rangle$, where T is the system lifetime; Sv is the state vector; VP is the sequence of variable ranges; CP is the sequence of variable class relations, $\langle Cp_1, \dots, Cp_n \rangle$; IR is the sequence of invariant relations, $\langle Ir_1, \dots, Ir_m \rangle$ and HR is the sequence of history relations, $\langle Hr_1, \dots, Hr_k \rangle$.

In the following sections, for each relation, I will discuss the motivation for introducing it and the formal semantics of the relation. Examples of the relations will be given by illustrating how they can be used to specify properties of the reaction vessel.

3.5.1. Variable Class Relations

In the verification of specifications which are defined over a history set it may be necessary to assume certain properties of the restricted history functions, representing the behaviour of a variable during the system lifetime. To capture these properties constraints are imposed over the functions, by placing them into classes which have the same mathematical properties. These properties are defined as variable class relations. (It should be noted that the variable ranges can be considered as a very simple class relation.)

Definition: Variable Class relations

A variable class relation for a variable p_i (denoted by Cp_i) is a predicate built using standard relations and logical connectives, and one free function variable p_i . No other free variable may be used.

Semantics

The semantics of a variable class relation will be defined in terms of the satisfaction condition. We will say that a history H satisfies a variable class relation Cp_i if and only if the substitution of the function $H.p_i$ for the function p_i results in a well-defined expression which evaluates to true. This satisfaction will be denoted by $H \text{ sat } Cp_i$.

Although all possible class relations cannot be covered here, some of the more common ones (which should be adequate for most systems) will be discussed.

Data Variables

These are variables which possess the property that if their value changes they must pass through a sequence of values which includes all intermediate values. A typical example is a variable representing a simple selector on the operator console.

Definition: Data variables

Data variables possess the property that when their value changes from, say, K_0 to K_1 from time points t_0 to t_1 respectively, then for all values k in the range K_0 to K_1 there must be a time point during the interval $[t_0, t_1]$ when the variable takes the value k .

More precisely, if p_i is a data variable then

$\forall t_0, t_1 \in T: [(t_0 \leq t_1) \wedge (p_i(t_0) = K_0) \wedge (p_i(t_1) = K_1) \Rightarrow \forall k \in K: \exists t \in [t_0, t_1]: p_i(t) = k]$,
where $K = \{k \in V_{p_i}: K_0 \leq k \leq K_1\}$.

This property is a “generalisation” of the intermediate value theorem (for continuous functions), it extends the theorem to include discrete functions. To check the classification for a variable the ordering relation (\leq) must be defined over the values in the range set.

Non-decreasing Variables

These are variables which represent quantities that can never decrease during the lifetime of the plant. A typical example is a variable representing the total energy output from a plant since it was started up.

Definition: Non-decreasing variables

A non-decreasing variable is a variable which can never decrease in value as time increases.

More precisely, if p_i is a non-decreasing variable then $\forall \text{Int} \in \text{SI}(T): p_i(s(\text{Int})) \leq p_i(e(\text{Int}))$.

Similarly we can have the non-increasing, increasing and decreasing variables. To check the classification for a variable the ordering relation (\leq) must be defined over the values in the range set.

For example in the reaction vessel, the variable p_{20} (explosion) is non-decreasing.

Continuous Variables

The class of continuous variables includes most physical variables. They are those variables for which the restricted history function is continuous during the systems lifetime. Typical examples are the volume of liquid in a vessel and the flow rate of liquid into a vessel.

Definition: Continuous variables

For a continuous variable p_i the corresponding restricted history function $H.p_i$ is a continuous function over the systems lifetime.

More precisely, for a continuous variable p_i the following three conditions hold:

$$\forall t \in (s(T), e(T)): \lim_{x \rightarrow t} p_i(x) \text{ exists;}$$

$$\forall t \in (s(T), e(T)): \lim_{x \rightarrow t} p_i(x) = H.p_i(t);$$

$$\lim_{x \rightarrow s(T)^+} H.p_i(x) = H.p_i(s(T)) \wedge \lim_{x \rightarrow e(T)^-} p_i(x) = p_i(e(T)).$$

For example, of the real world variables of the reaction vessel the following are continuous variables: p_1, p_6, \dots, p_{16} . (Continuous variables are, of course, integrable over the closed interval T .)

Free Variables

The class of free variables, are those variables for which the null constraint is imposed on the restricted history functions. Typical examples are switches at the operator console.

Definition: Free variables

For a free variable the class relation is true.

For example, of the real world variables of the reaction vessel the following are free variables: $p_2, p_3, p_4, p_5, p_{17}, p_{18}, p_{19}$.

We will say that a history H of a system satisfies the variable class relations of a system if and only if the function H sat Cp_i , for all $i \in \{1, \dots, n\}$. This will be denoted by H sat CP .

For a particular application it may be necessary to specify additional classes. For example, if fluid mechanics are used to reason about the environment then a class of differentiable variables would be required. From the above examples it should be obvious

how to define a class of variables. From the definition of a variable class relation, it must be possible to determine whether or not a history satisfies the restrictions imposed by the class.

Class Table

The category and classes of the variables of a system can be represented concisely in a tabular format – as a class table. A class table has four columns; the headings and a brief description of the content of each column is given below.

- i) *Variables*: specifies the variables.
- ii) *Category*: specifies the category of the variables.
- iii) *Class*: specifies the class of the variables.
- iv) *Comments*: briefly justifies the category and class of the variables.

The categories and the classes of the reaction vessel are given in table 3.3.

Table 3.3: Class and Categories of Reaction Vessel Variables

Variables	Category	Class	Comments
p1	Physical	Perfect clock	This variable represents the perfect clock of the system (see later).
p2, p3, p4, p5, p17, p18, p19	Operator	Free	No restrictions are imposed on the histories of the variables which represent quantities on the operator interface.
p6, ..., p16	Physical	Continuous	The variables which represent the physical properties of the reaction vessel have continuous histories.
p20	Physical	Catastrophe	Once an explosion has occurred in a history the explosion variable remains true for the rest of the history. (see chapter 5)
p21, p22, p24, p28, p29	Controller	Free	No restrictions are imposed on the histories of the values of the lock, thermometer or regulator.
p23, p25, p26, p27	Controller	Continuous	These variables represents the extent to which a valve is opened, hence they are all continuous variables.

3.5.2. Invariant Relations

In a system state the possible values of a state variable will be constrained by relationships with each other. Invariant relations are used to concisely express relationships which hold during the entire system lifetime. These relations are formulated as system predicates.

Definition: System predicate

A system predicate (SysPred) is any predicate built using standard logical connectives; standard mathematical functions and relations; and n free value variables p_1, \dots, p_n of type Vp_1, \dots, Vp_n . No other free variables can be used.

Semantics

The semantics of a system predicate will be defined in terms of the satisfaction condition. A state value $V (= \langle x_1, \dots, x_n \rangle$, where x_i is of type Vp_i) satisfies a system predicate if and only if substitution of each x_i for p_i within the system predicate results in a well-defined Boolean expression which evaluates to true. (The evaluation of such an expression is done in the usual way.) We denote this by writing: $V \text{ sat SysPred}$.

Definition: Invariant relation

A system predicate SysPred is an invariant relation for a history H iff $\langle H.p_1(t), \dots, H.p_n(t) \rangle \text{ sat SysPred}$ for all $t \in T$. This satisfaction will be denoted by $H \text{ sat SysPred}$.

Real World Invariant Relations

Two examples of real world invariant relations are presented below, where the first property is derived from a physical law, and the second property is derived from the construction of the machinery.

- i) In a system which contains N moles of an ideal gas the variables which represent the pressure (p_x), volume (p_y) and temperature (p_z) of the gas must always satisfy the ideal gas law. This can be precisely stated as a simple invariant relation: $(p_x.p_y) = N.R.p_z$.
- ii) In the description of the reaction vessel an invariant relation is used to describe the relationship between the flow rates into a vessel. That is, the flow rate (p_9) into the vessel is the sum of FlowA (p_7) and FlowB (p_8). This can be precisely stated as: $p_9 = p_7 + p_8$.

Controller Invariant Relations

Invariant relations can be used to state simple relationships between sensors, actuators and the physical process. An example of how invariant relations can be used to specify the relationship between the actuators of a controller and the physical properties of a plant, drawn from the reaction vessel, is presented next. In the reaction vessel the flow

rate out of the outlet valve (p_{14}) is given by a function over the volume of liquid (p_{13}) in the vessel and the outlet valve setting (p_{27}): $p_{14} = f_{OC}(p_{13}, p_{27})$, where $f_{OC}(p_{13}, p_{27})$ gives the flow rate as a function of the valve setting and the volume of liquid in the vessel.

Two examples of how invariant relations can capture special properties of safety-critical systems (those of imprecision and redundancy) are discussed in the following paragraphs.

Imprecision

In process control systems, the value of a continuously changing variable is sampled only at discrete time points determined by a sampling interval. Thus the “measured” values are (at best) the “real” values at these time instances. However, even at the time point at which the system senses the real value the measured value can only be an approximation of the real value – due to measurement errors. From the above discussion we can infer that it is unlikely that the real variables and the measured variables will be equal during a history. Nevertheless, assuming that the sensor is working as required, the real variable (p_i) will always be related to the measured value (p_j); the relationship should be expressible in the form of a bound on the difference in the two values (i.e., the imprecision). Such relationships can be expressed simply by an invariant relation.

For example, in the reaction vessel, suppose the temperature (p_6) is measured by a thermometer (p_{24}). The two sources of imprecision are i) the maximum error in the thermometer, which is say ΔTp_1 , and ii) the maximum error due to discrete sampling, which is say ΔTp_2 (for a given sampling frequency). By combining the two error constants the following invariant relation can be defined: $|p_6 - p_{24}| \leq \Delta Tp_1 + \Delta Tp_2$. Of course, if we define $\Delta Tp = \Delta Tp_1 + \Delta Tp_2$, the invariant can be stated concisely as: $|p_6 - p_{24}| \leq \Delta Tp$.

Redundancy

In many safety-critical systems redundancy is employed at the component level, to achieve the required high level of reliability. Therefore it is reasonable to expect that a model for safety-critical systems should be able to represent redundant systems simply. In

particular the modification of a non-redundant description to a description with redundant components should be straightforward.

As an example of how invariant relations can be used to specify some issues of redundancy, the introduction of redundant sensors in the reaction vessel is considered below. Suppose that in the safety controller specification a condition is imposed on the thermometer (p_{24}) in the form $p_{24} \geq \text{temp}_3$, which is a trigger to a recovery event, that is:

If $p_{24} \geq \text{temp}_3$ Then *recovery actions*

Further, let us suppose that the safety controller analysts decide that the single thermometer test is too unreliable, and he wishes to improve the probability that the recovery actions are initiated by adding a redundant thermometer (say, p_{30}). The component description can be simply modified by replacing the invariant relation over the thermometer (p_{24}) by the following invariant relation: $|p_{24} - p_6| \leq \Delta T_p \vee |p_{30} - p_6| \leq \Delta T_p$. The confidence which can be placed in the second invariant relation being much greater (i.e. the probability that the invariant holds for an arbitrary history of the system). The trigger condition can then be restated as $\text{Min}(p_{24}, p_{30}) \geq \text{temp}_3$, that is:

If $\text{Min}(p_{24}, p_{30}) \geq \text{temp}_3$ Then *recovery actions*.

3.5.3. History Relations

Some of the relationships over state variables cannot be expressed using invariant relations. A typical example is the relationship between the change in the liquid level over a given interval of time and the flow rate into and out of the vessel during the interval. The main reason why these sorts of relationships cannot be specified by invariant relations is that the relationship, between the variables, must be imposed over all evolutions (partial histories) of a history; whereas invariant relations can only impose constraints over all the states of a history. For relationships which hold for all evolutions a more powerful relation is required – this is referred to as a history relation; these relations are formulated as history predicates.

Definition: History predicate

A history predicate is a predicate which is built using standard mathematical functions, relations and logical connectives; and two free time variables T_0, T_1 , $2.n$ free value variables $p_{1,0}, \dots, p_{n,0}, p_{1,1}, \dots, p_{n,1}$ (where $p_{i,j}$ has type Vp_i), n free function variables p_1, \dots, p_n (where p_i is a function of class Cp_i). No other free variables may be used.

Semantics

We will say that a history H satisfies a history predicate HistPred for an interval (open or closed) Int if and only if the expression resulting from substituting: i) $s(\text{Int})$ for T_0 , ii) $e(\text{Int})$ for T_1 , iii) $H.p_i(s(\text{Int}))$ for $p_{i,0}$, for all i , iv) $H.p_i(e(\text{Int}))$ for $p_{i,1}$ for all i , v) $H.p_i$ for p_i for all i , is a well-defined expression which evaluates to true. We will denote this satisfaction by writing: $H \text{ sat HistPred@Int}$.

Definition: History relation

A history predicate HistPred is a history relation for a history H if and only if $H \text{ sat HistPred@Int}$ for all $\text{Int} \in SI(T)$. This satisfaction will be denoted by $H \text{ sat HistPred}$.

Real World History Relations

Two examples of history relations, drawn from the history description of the reaction vessel, are discussed below.

Temperature relation

Let us suppose that an analysis of the derivative of the temperature (p_6), shows that the maximum rise in the temperature per second is bounded by ΔT_m . This can be formally expressed by the following history relation: $p_{6,1} \leq p_{6,0} + \Delta T_m \times (T_1 - T_0)$. To simplify history relations which are expressed in terms of the duration of an interval, an abbreviation convention is adopted in which *dur* represents $T_1 - T_0$ in a history relation. The abbreviated version of this relation is: $p_{6,1} \leq p_{6,0} + \Delta T_m \times \text{dur}$.

Volume relation

For the reaction vessel the volume of liquid in the vessel (p_{13}) at the end of any interval Int is the volume at the start point of Int minus the integral of the flow rate out of the vessel during Int , provided the flow rate into the vessel is zero during Int . To simplify the history

relation which describes this property, an abbreviation convention is adopted in which the limits for an integral over the interval are omitted. The (abbreviated) history relation for the volume relation is: $(\forall t: p_9(t)=0) \Rightarrow (p_{13,1} = p_{13,0} - \int p_{16}(t) dt)$.

Controller History Relations

History relations can be used to specify the relationship between complex sensors, actuators and the physical process, typical examples are automatic control devices. An investigation into the construction of several specifications, showed that they can become involved and difficult to understand. Though the abbreviations discussed in the previous section can simplify some expressions, the specification of relationships between transducers and the physical process can still lead to complicated history relations. To further simplify the expressions an abbreviation convention is introduced. If the universal quantifier is imposed over a variable t on the interval $[T_0, T_1]$, the interval will be omitted. Furthermore if the universal quantifier is imposed over the variable t on the interval $[T_0 + \Delta x, T_1 - \Delta y]$, the interval will be denoted as $(\Delta x, -\Delta y)$, both Δx and Δy are non-negative.

As an example of how the abbreviations can be used to simplify history relations, the history relation for the temperature regulator of the reaction vessel is given below in full (i) and in its abbreviated form (ii):

- i) $\forall t \in [T_0, T_1]: [(p_{28}(t) = \text{on} \wedge p_{29}(t) = p_{29,0} \wedge p_9(t)=0 \wedge T_1 - T_0 > \Delta Rt(p_{29,0})$
 $\Rightarrow \forall t \in [T_0 + \Delta Rt(p_{29,0}), T_1]: |p_{29} - p_6| \leq \Delta Reg];$
- ii) $\forall t: (p_{28}(t) = \text{on} \wedge p_{29}(t) = p_{29,0} \wedge p_9(t)=0 \wedge \text{dur} > \Delta Rt(p_{29,0}))$
 $\Rightarrow \forall t: (\Delta Rt(p_{29,0}), 0): |p_{29} - p_6| \leq \Delta Reg).$

The history relation of the temperature regulator, states that if the regulator is on (i.e., $p_{28} = \text{on}$) for an interval during which regulator set (p_{29}) is constant, the flow rate out of the vessel is zero; and the duration of the interval is greater than the value given by $\Delta Rt(p_{29,0})$, then after the first $\Delta Rt(p_{29,0})$ seconds of the interval the temperature is within ΔReg of the set value. Hence the history relation captures the fact that ΔRt is a function which defines

an upper bound on the time taken for the regulator (under the given circumstances) to stabilize the temperature of the reaction vessel at approximately the set value (i.e., $p_{29,0}$).

Relations Table

The relations of a system can be concisely represented in a tabular format – as a relations table. For each relation of the environment description there is a row in the relations table. The table has four columns; the headings and a brief description of the content of each column is given below.

- i) *Number (No.)*: specifies the number of the relation.
- ii) *Related variables*: specifies the variables over which the relation is defined.
- iii) *Relationships*: specifies the relation in the standard notation.
- iv) *Comments*: briefly justifies the relation.

For example, an extract of a relations table over the real world variables of the reaction vessel is given in table 3.4.

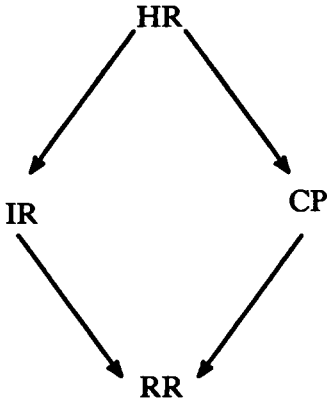
Table 3.4: Reaction Vessel Description Relations

No.	Related variables	Relationship	Comments
Ir ₁	p ₇ , p ₈ , p ₉	$p_9 = p_7 + p_8$	The flow rate into the vessel is the sum of FlowA and FlowB.
Ir ₂	p ₁₀ , p ₁₁ , p ₁₂ , p ₁₃	$p_{13} = p_{10} + p_{11} + p_{12}$	The volume of liquid in the vessel is the sum of the volumes of A, B and C.
Ir ₃	p ₁₄ , p ₁₅ , p ₁₆	$p_{16} = p_{15} + p_{14}$	The flow rate out of the vessel is the sum of OutflowC and OutflowD.

3.5.4. Comparison of Relations

In this section, I will briefly compare the sort of properties that can be expressed by the description relations and the variable ranges (for the purpose of this comparison it is convenient to consider variable ranges as range relations RR). Firstly, it should be stated that it is not being claimed that all possible properties can be represented by the description relation, only that a useful set of properties can be represented by the relations. Clearly, all the relations can be used to express the range of the variables. Furthermore, properties expressed as variable class relations and invariant relations can be expressed as

history relations. However, there is no simple relationship between the properties which can be expressed by the variable class and invariant relations. Hence, there exists a simple partial ordering of the expressiveness of the relations. This partial ordering is illustrated by figure 3.2.



An arc exists from a sort of relations A to another sort B if and only if any property expressed by a relation of sort B can be expressed by a relation of sort A.

Figure 3.2. A Partial Ordering of Description Relations and Ranges Relations

3.5.5. History Description Sets

A history description can be used to define a set of system histories which satisfy the restrictions imposed by the relations of the description – as a history description set.

Definition: History Description Set

The history set of a description Desc is the subset of all universal histories which satisfy the relations of the description. This will be denoted by Set(Desc).

$$\text{Set}(\text{Desc}) = \{H \in \Gamma H \mid H \text{ sat } CP \wedge H \text{ sat } C(IR) \wedge H \text{ sat } C(HR)\},$$

where $C(IR) = Ir_1 \wedge \dots \wedge Ir_m$ and $C(HR) = Hr_1 \wedge \dots \wedge Hr_k$.

A super set of the history description set can be defined if the restrictions imposed on the universal history set are restricted to those specified by the invariant relations. This set of histories will be referred to as the invariant histories.

Definition: Invariant histories

The invariant history set of an description Desc is the set of all universal histories which satisfy

the restrictions imposed by the invariant relations. This will be denoted by $Iset(Desc)$.

More precisely, $Iset(Desc) = \{H \in \Gamma H \mid H \text{ sat } C(IR)\}$.

3.6. Clocks

In the formal model clocks will be considered as entities which measure the absolute time of the time base during the operation of the system (i.e., the system lifetime). Clocks will be represented as state variables. We will mainly be concerned with two classes of clocks, those at the real world level (perfect clocks) and those at the controller level (controller clocks).

Perfect Clocks

A perfect clock is a hypothetical clock which can be used to specify constraints over the behaviour of the system at the real world level. In terms of the model a perfect clock is a physical variable.

Definition: Perfect clock

A physical variable is a perfect clock for a description D if and only if its variable class relation imposes the constraint that at any time point during the system lifetime the value of the variable is equal to the time point.

More precisely, $p_i \in P$ is a perfect clock for a description D iff $Cp_i(D) = (\forall t \in T: p_i = t)$.

Controller Clocks

A controller clock is a mechanical clock which can be used to specify constraints over the behaviour of the system at the controller level. A controller clock can be viewed as a “sensor” for a perfect clock; hence a controller clock is a controller variable. Several types of controller clocks can be defined, but here I will define a clock for which there is a bound on the difference between its value and the value of a perfect clock. These types of clocks will be called *bounded clocks*.

Definition: Bounded clocks

A controller variable is a bounded clock for a description D if and only if there is an invariant relation which imposes an upper bound on the difference in the value of the variable and a

perfect clock.

More precisely, $p_i \in C$ is a bounded clock for a description D iff there is an invariant relation in $IR(D)$ which is equivalent to $(|p_i - p_1| \leq \Delta v)$, where p_1 is a perfect clock.

3.7. Real-time Satisfaction Conditions

In this subsection I will introduce a set of satisfaction conditions that can be used to specify the behaviour of the system at the real world and controller levels. It is not claimed that all possible system behaviours can be specified by the real-time conditions, but for many systems in the class of systems investigated in this theses they should be sufficient to specify the required behaviour. In part this is justified by the examples presented in this section. (Further justification is given by the specifications that are expressed in chapters 4, 5 and 6; and the appendices B and C.) In the following subsections, I will introduce the satisfaction conditions, and give some examples of how they can be used to formally express informal specifications. The term constrained histories will be used as a general term for the set of description histories (DH) which satisfy these conditions; the set will be denoted by CH.

3.7.1. System Predicates

The first three conditions will be formulated as system predicates (see section 3.5.1). As an example, consider the following system predicate for a system with four variables: $\text{SysPred} = p_1 < p_2 \wedge p_3 \geq p_4$. Now, if we consider the two state values $V_1 = \langle 4, 5, 4, 3 \rangle$ and $V_2 = \langle 4, 5, 3, 4 \rangle$, we can say $V_1 \text{ sat SysPred}$ and $V_2 \text{ sat } \neg \text{SysPred}$.

Properties

In the formalisms below V stands for an arbitrary state value.

Negation: $\neg(V \text{ sat SysPred}) \Leftrightarrow V \text{ sat } \neg \text{SysPred}$.

Implication: $(\text{SysPred}_i \Rightarrow \text{SysPred}_j) \Leftrightarrow^{\text{def}} (\forall V \in \Gamma: [V \text{ sat SysPred}_i \Rightarrow V \text{ sat SysPred}_j])$.

Equivalence: $(\text{SysPred}_i \Leftrightarrow \text{SysPred}_j) \Leftrightarrow^{\text{def}} (\forall V \in \Gamma: [V \text{ sat SysPred}_i \Leftrightarrow V \text{ sat SysPred}_j])$.

Distributive (\wedge): $(V \text{ sat SysPred}_i \wedge V \text{ sat SysPred}_j) \Leftrightarrow (V \text{ sat SysPred}_i \wedge \text{SysPred}_j)$.

Distributive (\vee): $(V \text{ sat SysPred}_i \vee V \text{ sat SysPred}_j) \Leftrightarrow (V \text{ sat SysPred}_i \vee \text{SysPred}_j)$.

System predicates will not be used directly to construct specifications, but will be used to construct real-time satisfaction conditions which can then be used to impose constraints over histories.

System Predicate Examples

Some examples of how system predicates can be used to express conditions over the state of the reaction vessel are given below.

Temperature Predicate

The condition that *the temperature of the vessel is less than the minimum activation temperature*, is expressed by the following system predicate: $p_6 < \text{Mact}$.

Volume Predicate

The condition that *the volume of A is approximately (i.e., within ΔvA) of the set point value and the volume of B is approximately (i.e., within ΔvB) of the set point value*, is expressed by the following system predicate: $|p_{10} - p_2| \leq \Delta vA \wedge |p_{11} - p_3| \leq \Delta vB$.

If a system predicate is used in several constructs during the specification of a system, to simplify the references to the predicate, an abbreviation can be used to denote it. For example, the volume predicate of the reaction vessel is abbreviated to Dvol , where $\text{Dvol} = (|p_{10} - p_2| \leq \Delta vA \wedge |p_{11} - p_3| \leq \Delta vB)$.

3.7.2. Point Satisfaction

This is used to capture the notion of a system predicate being satisfied by the state given by evaluating a history at a specific time point. That is, point constraints are used to impose constraints over snapshots of a history.

Definition: Point satisfaction

A system predicate SysPred is satisfied for a history H at a time point t if and only if the state value $\langle H.p_1(t), \dots, H.p_n(t) \rangle \text{ sat } \text{SysPred}$. This is denoted by $H \text{ sat } \text{SysPred}@t$.

Properties

In the properties below H will stand for an arbitrary history and t an arbitrary time point.

Negation: $\neg(H \text{ sat } \text{SysPred}@t) \Leftrightarrow H \text{ sat } \neg \text{SysPred}@t$.

Implication: $(\text{SysPred}_i \Rightarrow \text{SysPred}_j)$

$$\Leftrightarrow (\forall H \in \Gamma H, \forall t \in T: [H \text{ sat SysPred}_i@t \Rightarrow H \text{ sat SysPred}_j@t]).$$

Equivalence: $(\text{SysPred}_i \Leftrightarrow \text{SysPred}_j)$

$$\Leftrightarrow (\forall H \in \Gamma H, \forall t \in T: [H \text{ sat SysPred}_i@t \Leftrightarrow H \text{ sat SysPred}_j@t]).$$

Distributive (\wedge): $(H \text{ sat SysPred}_i@t \wedge H \text{ sat SysPred}_j@t)$

$$\Leftrightarrow (H \text{ sat } (\text{SysPred}_i \wedge \text{SysPred}_j)@t]$$

Distributive (\vee): $H \text{ sat SysPred}_i@t \vee H \text{ sat SysPred}_j@t \Leftrightarrow H \text{ sat } (\text{SysPred}_i \vee \text{SysPred}_j)@t.$

Point Satisfaction Examples

Two useful roles that point satisfaction can play in specifying a set of constrained histories are discussed below.

At a specific time

Point satisfaction can be used as a quantitative temporal predicate to specify a set of histories for which a system predicate holds at a specified time point. For example, suppose we have the requirement that: “the temperature of the mixture in the vessel (p_6) should be below a specified value (temp) at a specified time (t_s)”. The set of histories which satisfy the requirement can be stated using by a point satisfaction condition as:

$$CH = \{H \in DH \mid H \text{ sat } (p_6 < \text{temp})@t_s\}.$$

At some time

Point satisfaction can also be used as a qualitative (c.f. linear temporal logic [Emer86]) temporal predicate to specify requirements which must hold at some time point. For example, suppose we have the requirement that: “the temperature of the liquid in the vessel (p_6) should be below a specified value (temp) at some time during the lifetime of the system”. The set of histories which satisfy the requirement can be stated using by a point satisfaction condition as: $CH = \{H \in DH \mid \exists t \in T: H \text{ sat } (p_6 < \text{temp})@t\}.$

3.7.3. Interval Satisfaction

This is used to capture the notion of a system predicate being satisfied by the set of states given by evaluating a history at each time point of a specific interval. That is interval

satisfaction is used to impose a constraint over evolutions of a history.

Definition: Interval satisfaction

A system predicate SysPred is satisfied for a History H during an interval Int if and only if $H \text{ sat SysPred}@t$ for all $t \in \text{Int}$. This is denoted by $H \text{ sat SysPred}@Int$.

Properties

In the properties below H is an arbitrary history and $\text{Int}_0, \text{Int}_1$ are two arbitrary intervals.

Negation: $\neg(H \text{ sat SysPred}_i@\text{Int}_0) \Leftrightarrow \exists t \in \text{Int}_0: H \text{ sat } \neg\text{SysPred}_i@t.$

Implication: $(\text{SysPred}_i \Rightarrow \text{SysPred}_j) \Leftrightarrow (\forall H \in \Gamma H, \forall \text{Int}_0 \in \text{SI}(T): [H \text{ sat SysPred}_i@\text{Int}_0 \Rightarrow H \text{ sat SysPred}_j@\text{Int}_0])$

Equivalence: $(\text{SysPred}_i \Leftrightarrow \text{SysPred}_j) \Leftrightarrow (\forall H \in \Gamma H, \forall \text{Int}_0 \in \text{SI}(T): [H \text{ sat SysPred}_i@\text{Int}_0 \Leftrightarrow H \text{ sat SysPred}_j@\text{Int}_0]).$

Distributive (\wedge): $H \text{ sat SysPred}_i@\text{Int}_0 \wedge H \text{ sat SysPred}_j@\text{Int}_0 \Rightarrow H \text{ sat } (\text{SysPred}_i \wedge \text{SysPred}_j)@\text{Int}_0.$

Distributive (\vee): $H \text{ sat SysPred}_i@\text{Int}_0 \vee H \text{ sat SysPred}_j@\text{Int}_0 \Rightarrow H \text{ sat } (\text{SysPred}_i \vee \text{SysPred}_j)@\text{Int}_0.$

Projection (\wedge, \cap): $H \text{ sat SysPred}_i@\text{Int}_0 \wedge H \text{ sat SysPred}_j@\text{Int}_1 \wedge (\text{Int}_0 \cap \text{Int}_1) \neq \emptyset \Rightarrow H \text{ sat } (\text{SysPred}_i \wedge \text{SysPred}_j)@(\text{Int}_0 \cap \text{Int}_1).$

Projection (\vee, \cap): $H \text{ sat SysPred}_i@\text{Int}_0 \vee H \text{ sat SysPred}_j@\text{Int}_1 \wedge (\text{Int}_0 \cap \text{Int}_1) \neq \emptyset \Rightarrow H \text{ sat } (\text{SysPred}_i \vee \text{SysPred}_j)@(\text{Int}_0 \cap \text{Int}_1).$

[Note: the condition $(\text{Int}_0 \cap \text{Int}_1) \neq \emptyset$ ensures that $\text{Int}_0 \cap \text{Int}_1$ is an interval (see lemma 3.1).]

Projection (\wedge, \cup): $H \text{ sat SysPred}_i@\text{Int}_0 \wedge H \text{ sat SysPred}_j@\text{Int}_1 \wedge (\text{Int}_0 \cap \text{Int}_1) \neq \emptyset \Rightarrow H \text{ sat SysPred}_i@(\text{Int}_0 \cup \text{Int}_1).$

[Note: the condition $(\text{Int}_0 \cap \text{Int}_1) \neq \emptyset$ ensures that $\text{Int}_0 \cup \text{Int}_1$ is an interval (see lemma 3.2).]

Interval Satisfaction Examples

Two useful roles that interval satisfaction can play in the specification of a set of constrained histories, will be discussed below (c.f. point satisfaction roles).

During a specific interval

Interval satisfaction can be used as a quantitative temporal predicate to specify a set of

histories for which a system predicate must hold in a specified interval. For example, suppose we have the requirement that: “the temperature of the liquid in the vessel (p_6) should be below a specified value (temp) during a specified closed interval given by $[t_r, t_s]$ ”.

The set of histories which satisfy the requirement can be specified as:

$$CH = \{H \in DH \mid H \text{ sat } (p_6 < \text{temp})@[t_r, t_s]\}.$$

During some interval

Interval satisfaction can be used as a qualitative temporal predicate to specify a set of histories for which a system predicate must hold for some interval in the systems lifetime. For example, suppose we have the requirement that: “there must be some interval during the systems lifetime of duration at least Δt when the volume of liquid A (p_{10}) is approximately (within Δl of) VolA select (p_2)”. The set of histories which satisfy the requirement can be specified by an interval satisfaction condition as:

$$CH = \{H \in DH \mid \exists \text{Int} \in SI(T): \text{dur}(\text{Int}) \geq \Delta t \wedge H \text{ sat } (|p_{10} - p_2| \leq \Delta l)@\text{Int}\}.$$

3.7.4. Events

In the specification of real-time systems a useful notion has been that of an event, which can be described informally as a condition which holds at a particular point in time. Though the notion of an event has often been used in the specification of real-time systems a formal definition of an event is rarely given. In a paper by Heninger [Heni80] it is recognised that an occurrence of an event (which is specified by a condition) is dependent on the behaviour of the system prior to the time at which the condition holds. However, the notion is not formalized.

In a paper by Jahanian and Mok [Jah86] events are introduced informally as “An event serves as a temporal marker, i.e., the occurrence of an event marks a point in time which is of significance in describing the behaviour of the system.” Later in the paper they formally relate the notion of an event to time by the definition of an occurrence function. The occurrence function maps the i^{th} occurrence of an event onto a non-negative integer which represents the time of the i^{th} occurrence. In the treatment of events by Jahanian and Mok it is implicit that the condition which must hold true for the occurrence of the event

does not hold at the time point just before the value given by the occurrence function, this property is not formally expressed as a property of an event.

In the discussion which follows, an event is considered to be a time point at which a condition holds at the end of an interval during which the condition did not hold. Thus an event constitutes the transition of a condition becoming true after a period in which it was false.

Event Satisfaction

The notion of an event which occurs (for the first time in an interval) at the end point of an interval is captured in the formal framework. Such an event can be defined in terms of point satisfaction: an *event* characterized by the system predicate SysPred is satisfied for a history H on an interval Int if and only if SysPred is not satisfied for every time point up to the end point of the interval and is satisfied at the end point of the interval.

Definition: Event satisfaction

The event characterized by SysPred is satisfied for a history H on an interval Int if and only if $H \text{ sat } \neg \text{SysPred}@t$ for all $t \in \text{Int} - \{e(\text{Int})\} \wedge H \text{ sat } \text{SysPred}@e(\text{Int})$. This satisfaction will be denoted by $H \text{ sat } \text{SysPred} \odot \text{Int}$.

Properties

Negation: $\neg (H \text{ sat } \text{SysPred} \odot \text{Int})$
 $\Rightarrow \neg H \text{ sat } \neg \text{SysPred}@e(\text{Int}) \vee \neg H \text{ sat } \text{SysPred}@e(\text{Int})$.

Equivalence: $(\text{SysPred}_i \Leftrightarrow \text{SysPred}_j) \Leftrightarrow$
 $(\forall H \in \Gamma H, \forall \text{Int} \in \text{SI}(T): [H \text{ sat } \text{SysPred}_i \odot \text{Int} \Leftrightarrow H \text{ sat } \text{SysPred}_j \odot \text{Int}])$.

Distributive (\wedge): $H \text{ sat } \text{SysPred}_i \odot \text{Int} \wedge H \text{ sat } \text{SysPred}_j \odot \text{Int}$
 $\Rightarrow H \text{ sat } (\text{SysPred}_i \wedge \text{SysPred}_j) \odot \text{Int}$.

Lemma 3.3

If a history satisfies the negation of a system predicate during any Int interval then the real-time event which is built using the system predicate cannot be satisfied by the history for any interval x included in Int .

More precisely, $\forall \text{Int} \in \text{SI}(T): [H \text{ sat } \neg \text{SysPred}@e(\text{Int}) \Rightarrow \forall x \in \text{SI}(\text{Int}): \neg H \text{ sat } \text{SysPred} \odot x]$.

Proof

The proof is given by showing that if we assume that the event is satisfied for any interval included in Int we have a contradiction.

Assume $\exists x \in SI(Int): H \text{ sat SysPred} \odot x$

$\therefore \exists t \in Int: H \text{ sat SysPred}@t.$

$\therefore \neg(H \text{ sat } \neg\text{SysPred}@Int) \quad (\text{contradiction}).$

Event Satisfaction Examples

Two examples of how event satisfaction can be used to specify constraints over the behaviour of the system are given below.

For a specific interval

Event satisfaction can be used as a qualitative temporal predicate to specify a set of histories for which an event must hold for some interval in the system lifetime.

For example suppose we have a system which contains a lamp with two bulbs, one green and one red. For which p_y and p_z represent the states of the green and red bulbs respectively. For such a system suppose we have the requirement that: “the green bulb is on during the interval Int; and the red bulb is on at the end point of Int, but at no other time point during the interval”. In this case we must specify the event of turning the red bulb on at the end of the interval. The set of histories that satisfy the requirement can be formally specified as: $CH = \{H \in DH \mid H \text{ sat } p_y = \text{on}@Int \wedge H \text{ sat } p_z = \text{on} \odot Int\}.$

For some intervals

Event satisfaction can be used as a qualitative temporal predicate to specify a set of histories for which an event must hold for some interval in the systems lifetime.

For example suppose we have the requirement that: “... at the instant the volume of liquid in the vessel becomes greater than or equal to v_1 the Indicator must be at red. ...”. In this case we must specify the fact that the indicator is at red when the event that marks the points at which the volume is greater than v_1 occurs. The set of histories that satisfy the requirement can be formally specified as:

$CH = \{H \in DH \mid \forall Int: (H \text{ sat } p_{13} \geq v_1 \odot Int \wedge \text{dur}(Int) > 0) \Rightarrow H \text{ sat } p_{18} = r@e(Int)\}.$

3.7.5. Time Bound Constraints

Time bound constraints will be used to impose constraints over an interval in the lifetime of a system. Since time bound constraints impose constraints over intervals, they can be used to impose restrictions over a history set only when they are used in conjunction with system predicates.

Definition: Time bound constraint

A time bound constraint Tb is any predicate built using standard logical connectives; standard mathematical functions; and a free variable Iv of type SI(T). No other free variables can be used.

Semantics

The semantics of a time bound constraint will be defined in terms of the satisfaction condition. We will say that an interval Int satisfies a time bound constraint Tb if and only if the substitution of Int for Iv into Tb leads to a well-defined Boolean expression which evaluates to true. We will denote this by Int sat Tb .

Time Bound Constraint Examples

As was stated previously, by themselves, time bound constraints cannot impose any restrictions on the set of system histories. However, they can be used in conjunction with the point and interval constraints.

Bounded point satisfaction

Time bound constraints can be used to impose a bound on the time between which two point satisfaction conditions hold.

For example, suppose we have the requirement that: "... If the volume of liquid A (p_{10}) in the vessel is equal to v_0 at some time point (t_0), there must be a time point (t_1), within Δt of t_0 , at which the volume of liquid A in the vessel is equal to v_1 ". The set of histories which satisfy the above requirement can be given as:

$$\begin{aligned} \text{CH} &= \{H \in \text{DH} \mid (\forall t: (H \text{ sat } (p_{10} = v_0)@t) \\ &\Rightarrow \exists \text{Int}: s(\text{Int}) = t \wedge (H \text{ sat } (p_{10} = v_1)@e(\text{Int}) \wedge \text{Int sat dur(Iv)} \leq \Delta t)\}. \end{aligned}$$

Bounded interval constraints

Time bound constraints can be used to impose constraints on the duration of an interval, during which an interval satisfaction condition must hold.

For example, suppose we have the requirement that: “there must be some interval of duration at least Δt , during the system’s lifetime when the volume of liquid A (p_{10}) is approximately (within ΔvA of) the required volume (p_2)”. The set of histories which satisfy the above requirement can be given as:

$$CH = \{H \in DH \mid \exists \text{Int}: H \text{ sat } (|p_{10} - p_2| \leq \Delta vA) @ \text{Int} \wedge \text{Int sat dur}(\text{Iv}) \geq \Delta t\}.$$

3.7.6. Termination Predicate

For most systems it will be necessary to have a predicate that will be true when a system has shut down. This can be achieved by the construction of a special system predicate using a perfect clock, called the termination predicate. The termination predicate holds when the value of a perfect clock (say, p_i) is the end point of the system lifetime, i.e., $p_i = e(T)$; this predicate will be abbreviated to Ω .

Properties

When a termination predicate is used to specify a satisfaction condition the following properties can be inferred. In the following formalisms H represents an arbitrary history, t an arbitrary time point and Int an arbitrary interval.

- i) The termination predicate is *point satisfied* for a history H at a time point t if and only if the time point is the end point of the system lifetime (i.e., $H \text{ sat } \Omega @ t$ iff $t = e(T)$).
- ii) The termination predicate is *interval satisfied* for a history H during an interval Int if and only if the time point contains only the end point of the system lifetime (i.e., $H \text{ sat } \Omega @ \text{Int}$ iff $\text{Int} = \{e(T)\}$).
- iii) The *event* characterized by the termination predicate is satisfied on an interval Int if and only if the end point of the interval is the end point of the system lifetime (i.e., $H \text{ sat } \Omega \odot \text{Int}$ iff $e(\text{Int}) = T$).

Remark: Although, the termination predicate is defined over a physical variable it will be used to specify system behaviour at both the real world and controller levels.

3.8. Summary

The chapter introduced the basic concepts of the specification model. The chapter started with the representation of a base time set as a subset of the non-negative reals. The notion of a *time point* and *time interval* of a system were defined as an element and subset of the base time set, respectively. The *state vector* of a system was defined as a sequence of state variables of a system and for each state variable the notion of a *range set* was defined as the set of possible values for that variable. The *state space* of a system was defined as the cross product of the range sets of the state variables. The structure discussed in chapter two was reflected in the model by the definition of variables categories which are used to classify the state variables in terms of the properties represented, that is, real-world or controller (and safety-critical or mission-oriented).

The concept of system behaviour was formally defined in terms of *histories*, which are functions from the time base of a system to its state space. For a given system the *universal history set* was defined, which represents the set of all behaviours of the system. The restrictions imposed on system behaviour by the intrinsic properties of the environment were captured by imposing constraints over the universal history set. To specify the constraints the concept of a *history description* was introduced, which consists of three types of relations: *variable class relations*, *invariant relations* and *history relations*.

To specify the system behaviour required a set of real-time *satisfaction conditions* were introduced, and these were formulated in terms of system predicates. Several examples which illustrated how informal requirements can be represented in terms of the satisfaction conditions were discussed.

In conclusion, this chapter has introduced the concept of history descriptions as formal constructs which provide a means for the concise representation of the restrictions imposed on system behaviour by the environment at the real-world and controller levels. In addition, it was shown how history descriptions can be used to relate system behaviour at the two levels. It was shown, by presenting several examples, that the satisfaction conditions can be used to express formally, informal requirements. However, the resulting

specifications are quite complicated. If a full system specification was built from the satisfaction conditions in an undisciplined way, the resultant specification would soon become difficult to construct and check.

One approach to avoid badly structured and unreadable specifications, would be to develop a set of guidelines for a disciplined use of the satisfaction conditions for the expression of specifications. However, the absence of a structure to the constructs used to express the specifications stated in terms of the satisfaction conditions, would complicate the adoption of a suitable development methodology. An alternative approach to overcome the problems associated with the construction of system specifications from the satisfaction condition is to formulate a higher-level construct – specified in terms of the satisfaction conditions. Such a construct should lead to a structure to the system specifications and support the adoption of a development methodology. This second approach is followed in the next chapter which introduces the concept of a *mode* as the basic building block of complex specifications.

Chapter 4 - Mode Theory

Except for the most trivial systems, if a specification is constructed by the unrestricted use of the real-time satisfaction conditions (introduced in chapter three), it will consist of a complicated set of point, interval and event satisfaction conditions; and time bound constraints. Such a set of constraints would be both difficult to construct and check. It has been appreciated by some researchers that it may be necessary to impose an “artificial” structure on the specification, to simplify the analysis and synthesis of specifications [Hare86, Jaha88]. In fact, the necessity of a structured specification technique, was stressed as one of the essential attributes of a suitable model. To promote the development of a structured specification, a simple, but not too restrictive construct which can be used to structure specification is required. The construct which is proposed is a *mode*.

4.1. Modes

It is often observed that the behaviour exhibited by safety-critical computing systems (and other real-time systems) can be partitioned into intervals, during which the behaviour of the system depends on only a subset of the systems variables. Each of these intervals will be associated with a particular task (e.g., loading a vessel with a given chemical). Such tasks should be clearly defined, in particular there should be a precise statement of the conditions which exist when the task is completed. The intervals during which a system performs a specific task are often referred to as modes [Heni80, Jaha88]. It is proposed that modes (as defined overleaf) can be used to simplify the construction (and analysis) of formal specifications, in a similar way that procedures can be used to simplify the construction (and analysis) of implementations. It is not being suggested that modes are the best way to structure specifications, only that they are a useful method. The behaviour of a process control system is not characterized by a single task, rather as a set of tasks which must be performed in accordance to specific ordering constraints. It is proposed that the behaviour of each task be specified using modes, which define the behaviour of the system at the start of the task, during the task and the behaviour that must be exhibited by the system for the task to be completed. A graphical construct called a *mode graph* (see

section 4.3) will be used to specify the transitions between the modes of the tasks of a system, and the mode graphs of the tasks combined to produce a mode graph that specifies the behaviour of the system. The key feature of a mode is the application of system predicates to describe the behaviour of a task.

Definition: Mode

A mode is a five-tuple, $Mode = \langle Start, Inv, End, LB, UB \rangle$, where $Start$, Inv and End are system predicates, and LB and UB are time values (or time valued functions).

Semantics

The semantics will be defined using the **sat**, **@** and **⊙** notation.

$Mode = \langle Start, Inv, End, LB, UB \rangle$.

$H \text{ sat } Mode @ Int$ iff

$H \text{ sat } Start @ s(Int) \wedge H \text{ sat } Inv @ Int \wedge H \text{ sat } End \odot Int \wedge$

$Int \text{ sat } (dur(Iv) \geq LB \wedge dur(Iv) \leq UB)$.

Remark: When LB and UB are defined as functions, these functions are evaluated at the start point of the interval Int .

Discussion

The role of each component in a mode structure is described in detail below with reference to the semantics of a mode.

The system predicate *Start* is referred to as the *start predicate* of a mode and specifies the condition which must hold at the start of the mode. In the definition of the semantics of a mode, *Start* is used as the system predicate for a point satisfaction condition which is imposed at $s(Int)$ to stipulate that it holds at the start of a mode.

The system predicate *Inv* is referred to as the *invariant predicate* of the mode and specifies the conditions which must hold while the system is in that mode. In the definition of the semantics of a mode, *Inv* is used as the system predicate for an interval satisfaction condition which is imposed during the interval Int to stipulate that it holds throughout the lifetime of a mode.

The system predicate *End* is referred to as the *end predicate* of a mode and specifies the condition which must hold, for the first time since the system entered the mode, at the

instant the system leaves the mode. In the definition of the semantics of a mode, *End* is used as the system predicate for an event which is imposed on the interval *Int* to stipulate that it holds (for the first time in the interval) at the end of the interval.

The time value (or function) *LB* is referred to as the *lower bound* of the mode and imposes a lower bound on the lifetime of the mode. In the definition of the semantics of a mode *LB* is used as a constant for a time bound constraint imposed on $\text{dur}(\text{Int})$ which stipulates that the lifetime of the mode must not be less than *LB*. The time value (or function) *UB* is referred to as the *upper bound* of the mode and is used to impose an upper bound on the lifetime of the mode. In the definition of the semantics of a mode *UB* is used to specify a time bound constraint which stipulates that the lifetime of the mode must not exceed *UB*. If *UB* is a constant it must be non-zero, if it is a function zero must not be in its range. The time bound constraints constructed using *UB* and *LB* are collectively referred to as the *lifetime constraints* of a mode. If no constraint is required on the lower bound of the lifetime then *LB* is given the value zero; if no constraint is required on the upper bound then *UB* is set to the duration of the system lifetime (i.e., $\text{dur}(T)$).

Notation

In a general discussion on the properties of modes, a mode will be denoted by *m*, if more than one mode is used in an expression, they will be denoted as m_1, m_2, \dots . (When modes are used in the construction of specifications of a given system, a mode is given the name of the task it specifies.) When reasoning about the properties of modes, it will be necessary to refer to components of a mode. The following notation will be used: *Start(m)* represents the start constraint of the mode *m*, *Inv(m)* represents the invariant of the mode, *End(m)* represents the end constraint of the mode, *LB(m)* and *UB(m)* represent the lower bound and upper bound of a mode respectively. If the mode is obvious from the context the name of the mode is dropped, and *Start*, *Inv*, *End*, *LB* and *UB* are used.

Two corollaries (4.1 and 4.2) which follow from the definition of the semantics of a mode, are given below.

Corollary 4.1

If a history satisfies a mode for an interval then the invariant predicate and complement of the end predicate must hold prior to the end point of the interval.

More precisely, $\forall H \in \Gamma H: [H \text{ sat } m@Int \Rightarrow H \text{ sat } (Inv \wedge \neg End)@Int - \{e(Int)\}]$.

Proof. Immediate from the semantics of a mode.

Corollary 4.2

If a history satisfies a mode for an interval then the invariant predicate and the start predicate must hold at the start of the interval, and the invariant predicate and the end predicate must hold at the end of the interval.

More precisely,

$\forall H \in \Gamma H: [H \text{ sat } m@Int \Rightarrow H \text{ sat } (Start \wedge Inv)@s(Int) \wedge H \text{ sat } (Inv \wedge End)@e(Int)]$.

Proof. Immediate, from the semantics of a mode.

The concept of an *unified mode set* is introduced to give a precise definition of a set of modes which are specified over the same system. A set of modes is an unified mode set if and only if all the modes in the set are defined over the same universal history set.

4.1.1. History Graphs

The formal definition of the satisfaction of a mode gives a precise statement of the semantics of a mode. However, the definition does not provide an intuitive (visual) feeling of the behaviour represented by histories which satisfy the mode. To visualize the behaviour a line (which represents a history) can be labelled with the end points of the mode, and marked with the properties of a history that can be derived from the satisfaction of a mode. These properties are: i) the predicate *start* is satisfied at the start point of the mode; ii) the predicate *start* is satisfied during the lifetime of the mode; iii) the event *end* is satisfied on the lifetime and iv) the duration of the lifetime is within the time bounds. Such graphs, will be referred to as history graphs. Pictorial representations of history graphs are illustrated and discussed in figure 4.1. These graphs allow an easier understanding of the behaviour specified by several modes, in particular they can be used to illustrate proofs.

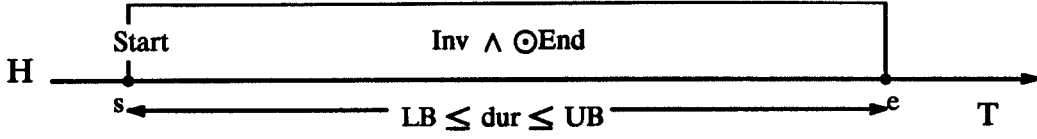


Figure 4.1. History Graphs

4.1.2. Mode Properties

In this section I will introduce three properties related to the satisfaction of a mode.

Definition: Start satisfaction

We will say that a history H starts a mode m during an interval Int if and only if the history satisfies the start predicate of the mode at the start of Int and the invariant predicate during Int ; and satisfies the negation of the end predicate during Int or the event given by the end predicate on Int . This will be denoted by $H \text{ sat } \mathfrak{f}(m)@Int$.

More precisely,

$$H \text{ sat } \mathfrak{f}(m)@Int \text{ iff } H \text{ sat } \text{Start}(m)@s(Int) \wedge H \text{ sat } \text{Inv}(m)@Int \wedge (H \text{ sat } \neg \text{End}(m)@Int \vee H \text{ sat } \text{End}(m)@Int).$$

Definition: End satisfaction

We will say that a history H ends a mode m during an interval Int if and only if the history satisfies the invariant predicate during Int and the event given by the end predicate on the interval. This will be denoted by $H \text{ sat } \mathfrak{g}(m)@Int$.

More precisely, $H \text{ sat } \mathfrak{g}(m)@Int \text{ iff } H \text{ sat } \text{Inv}(m)@Int \wedge H \text{ sat } \text{End}(m)@Int$.

Clearly both start satisfaction and end satisfaction are weaker than the satisfaction of a mode. However, under certain conditions the start and end satisfaction of a mode for a history can imply the satisfaction of the mode. The conditions are given in the lemma which follows.

Lemma 4.1

If a history H starts a mode during an interval Int_0 and ends the mode during another interval Int_1 , and the intervals intersect with the start (resp. end) point of Int_0 temporally preceding the start (resp. end) point of Int_1 , and the difference between the $s(Int_0)$ and $e(Int_1)$ is within the time bounds of the mode then the history satisfies the mode during the union of the two intervals.

More precisely, $\forall H \in \Gamma H: \forall \text{Int}_0, \text{Int}_1 \in \text{SI}(T)$:

$H \text{ sat } \mathcal{I}(m)@ \text{Int}_0 \wedge H \text{ sat } \mathcal{G}(m)@ \text{Int}_1 \wedge (\text{Int}_0 \cap \text{Int}_1 \neq \emptyset) \wedge s(\text{Int}_0) \leq s(\text{Int}_1) \wedge e(\text{Int}_0) \leq e(\text{Int}_1) \wedge$
 $\text{LB}(m) \leq e(\text{Int}_1) - s(\text{Int}_0) \leq \text{UB}(m) \Rightarrow H \text{ sat } m@ (\text{Int}_0 \cup \text{Int}_1).$

Proof. The proof for this lemma is illustrated by the history graphs in figure 4.2.

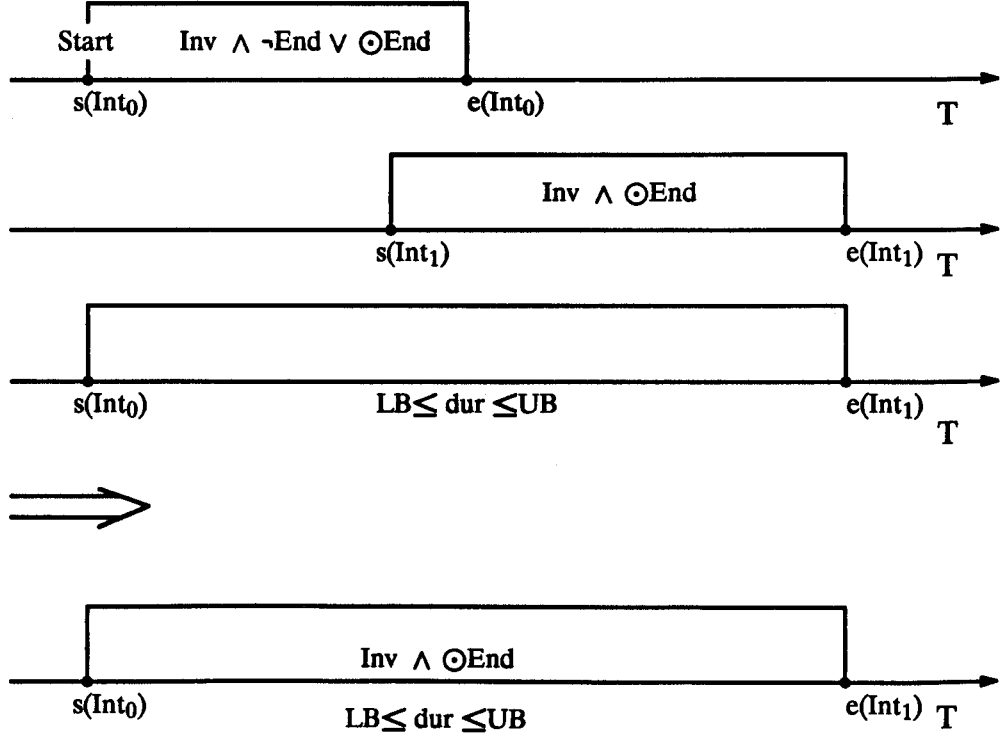


Figure 4.2. Graphical Argument for Lemma 4.1.

Mode Residence

To capture formally what is meant by when we say a history resides in a mode during an interval, the notion of mode residence is introduced.

Definition: Mode Residence

A system history H resides in a mode m during an interval Int if and only if Int is a subset of an interval for which the history satisfies the mode. This is denoted by $H \text{ res } m@ \text{Int}$.

More precisely, $H \text{ res } m@ \text{Int}_0$ iff $\exists \text{Int}_1 \in \text{SI}(T): H \text{ sat } m@ \text{Int}_1 \wedge \text{Int}_0 \subseteq \text{Int}_1$.

4.1.3. Mode Relationships

Two formal relationships will be defined for modes – the equivalence and implication relationships. Both relationships are dependent on the set of histories over which the satisfaction of the mode is expressed.

Mode Equivalence

The equivalence relationship between two modes will be defined in terms of the satisfaction condition and the histories over which the modes are imposed.

Definition: Mode equivalence

Two modes are equivalent for a set of histories HH if and only if all histories in the set which satisfy one mode during an interval satisfy the other mode during that interval.

More precisely, $m_1 \simeq^{HH} m_2$ iff $\forall H \in HH: \forall \text{Int} \in \text{SI}(T): [H \text{ sat } m_1 @ \text{Int} \Leftrightarrow H \text{ sat } m_2 @ \text{Int}]$.

The relation \simeq^{HH} is an equivalence relation. If two modes (say, m_1, m_2) satisfy the equivalence relationship for the universal history set of a system the relation is abbreviated to $m_1 \simeq m_2$.

Mode Implication

An implication relationship can be defined between two modes, in a way that is similar to the equivalence relation.

Definition: Mode implication

Mode m_1 implies mode m_2 for a set of histories HH if and only if all histories in the set which satisfy m_1 during an interval satisfy m_2 during that interval, in which case we say m_2 is a consequent of m_1 .

More precisely, $m_1 \sim^{HH} m_2$ iff $\forall H \in HH: \forall \text{Int} \in \text{SI}(T): [H \text{ sat } m_1 @ \text{Int} \Rightarrow H \text{ sat } m_2 @ \text{Int}]$.

Properties

The main properties of the implication relationship are stated below.

Reflexive: $m \sim^{HH} m$, for all $m \in \text{Mod}$.

$((m_1 \sim^{HH} m_2) \wedge (m_2 \sim^{HH} m_1)) \Rightarrow m_1 \simeq^{HH} m_2$, for all $m_1, m_2 \in \text{Mod}$.

Transitive: $((m_1 \approx^{HH} m_2) \wedge (m_2 \approx^{HH} m_3)) \Rightarrow m_1 \approx^{HH} m_3$, for all $m_1, m_2, m_3 \in \text{Mod}$.

Remark: If two modes (say, m_1, m_2) satisfy the implication relationship for the universal history set of a system the relation is abbreviated to $m_1 \approx m_2$.

Mode Example

In this section, I will discuss (by example) how a set of informal requirements over the behaviour of a system during an interval, can be expressed concisely as a mode. The example given is from the mission real world specification of the reaction vessel; it defines the collect mode which concerns the emptying of the vessel and the collection of the product.

Collect Mode

The system is in this mode while the product is being collected. During this mode Plant select must be at collect, the Temperature of the vessel must be within the activation temperature range and the Indicator must be at red, or the vessel must be empty, and the Indicator must be at red or green. The system must leave this mode as soon as the vessel is empty and the Indicator is at green. The system must spend at most ΔC seconds in this mode.

Collect = $\langle \text{true}, \text{Inv}, p_{13} = 0 \wedge p_{18} = g, 0, \Delta C \rangle$,

where Inv = $(p_5 = \text{collect} \wedge ((ST \wedge p_{18} = r) \vee p_{13} = 0) \wedge p_{18} \in \{g, r\})$, and

ST = $(0 \leq p_6 - p_4 \leq \Delta Tv)$.

4.1.4. Mode Categories

Modes will be divided into two disjoint categories: *unbounded* and *bounded*, based on the time bound constraints which are imposed on the satisfaction of a mode.

Unbounded Modes

The important property of an unbounded mode is that the satisfaction of the mode during an interval is independent of the duration of the interval. However, there is still a minimal constraint on the duration of a mode because the earliest start point is $s(T)$ and the latest end point is $e(T)$. Furthermore, for a given history it may be possible to derive a bound on the duration of the mode from the description relations and the system predicates of a mode.

Definition: Unbounded mode

An unbounded mode is a mode in which LB is equal to zero and UB is equal to the duration of the system lifetime (i.e., $dur(T)$).

An example of an unbounded mode from the mission specification of the reaction vessel is given below; it defines the set points mode which concerns the selection of the set points by the operator., 0, $dur(T)$

Set points Mode

The system is in this mode while the operator selects the required set points. At the start of the mode the plant select is at on. During the mode, Plant select must be at on or start, the vessel must be empty and the Indicator must be green. The system must leave this mode as soon as Plant select is turned to start.

Set points = $\langle p_5 = \text{on}, p_5 \in \{\text{on}, \text{start}\} \wedge p_{13} = 0 \wedge p_{18} = g, p_5 = \text{start} \rangle$.

To simplify the formal specifications of unbounded modes, the time values will be omitted, hence they are expressed as triples. Thus, the abbreviated version of the set points mode is as: Set points = $\langle p_5 = \text{on}, p_5 \in \{\text{on}, \text{start}\} \wedge p_{13} = 0 \wedge p_{18} = g, p_5 = \text{start} \rangle$.

Bounded Modes

The important property of a bounded mode is that the satisfaction of the mode during an interval is dependent on the duration of the interval.

Definition: Bounded mode

A bounded mode is a mode which is not unbounded.

An example of bounded mode from the mission specification of the reaction vessel is given below; it defines the set up mode which concerns the loading of the vessel with the required volumes of A and B.

Set up Mode

The system is in this mode while the required volumes of A and B are being loaded into the vessel. During this mode, Plant select must be at start, Temperature must be less than the minimum activation temperature or the vessel must be empty and the Indicator green. The system must leave this mode as soon as the volumes of A and B are both approximately equal to

their set point values. The system must spend at most $f_{ST}(p_2, p_3)$ seconds in the mode, where f_{ST} is a function which defines an upper bound on the time taken to fill the vessel with the required volumes.

Set up = $\langle \text{true}, p_5 = \text{start} \wedge (p_6 < \text{Mact} \vee p_{13} = 0) \wedge p_{18} = g, \text{Dvol}, 0, f_{ST}(p_2, p_3) \rangle$,
where $\text{Dvol} = (|p_{10} - p_2| \leq \Delta v_A \wedge |p_{11} - p_3| \leq \Delta v_B)$.

4.1.5. Mode Consistency

Some modes will be unsatisfiable for a given history description. There are two potential causes which can lead to a mode being unsatisfiable. The logic of a mode may be inconsistent – for example if the start predicate of a mode implies the negation of the invariant predicate then the mode is clearly unsatisfiable, for any possible history. The predicates of a mode may be in conflict with the relations of a history description, for example if an invariant relation of a history description implies the negation of the end predicate of a mode then the mode is unsatisfiable for that history description.

The satisfaction condition can be used to define a consistent mode for a given history description set, as a mode which is satisfiable in the invariant set of the description

Definition: Mode Consistency

A mode is consistent, for a set of invariant histories of a description D, if it is possible for at least one history of the set to satisfy the mode. This will be denoted by $m \text{ con } D$.

More precisely, $m \text{ con } D$ iff: $\exists H \in \text{Iset}(D): \exists \text{Int} \in \text{SI}(T): H \text{ sat } m@ \text{Int}$.

Remark. The above definition does not include restrictions placed on the histories by class and history relations.

Mode Consistency Checks

The consistency of a mode is checked by confirming the following two conditions.

i) *There exists a state vector that satisfies the conjunction of the start predicate, invariant predicate and negation of the end predicate; and the invariant relations.*

More precisely, $\exists V \in \Gamma(\text{SY}): V \text{ sat } \text{Start}(m) \wedge \text{Inv}(m) \wedge \neg \text{End}(m) \wedge C(\text{IR}(D))$.

ii) *There exists a state vector that satisfies the conjunction of the invariant and end predicate*

and invariant relations.

More precisely, $\exists V \in \Gamma(\text{SY}): V \text{ sat } \text{Inv}(m) \wedge \text{End}(m) \wedge C(\text{IR}(D))$.

Mode Consistency Theorem

The mode consistency theorem (theorem 4.1) shows that if the mode consistency checks (given above) are confirmed for a mode, against a given history description, then that mode is consistent for the history description.

Theorem 4.1

If there exists a state vector that satisfies the start predicate, invariant predicate and negation of the end predicate of a mode and the invariant relations of a history description; and there exists another state vector that satisfies the invariant predicate, end predicate of a mode and invariant relations of a mode, then there exists a history in the invariant set of the history description that satisfies the mode for an interval.

More precisely,

$$(\exists V \in \Gamma(\text{SY}): V \text{ sat } \text{Start}(m) \wedge \text{Inv}(m) \wedge \neg \text{End}(m) \wedge C(\text{IR}(D))) \wedge \\ (\exists V \in \Gamma(\text{SY}): V \text{ sat } \text{Inv}(m) \wedge \text{End}(m) \wedge C(\text{IR}(D))) \\ \Rightarrow \exists H \in \text{Iset}(\text{SY}): H \text{ sat } m@Int.$$

Proof.

The following two lemmas are required in a proof of the theorem.

Lemma 4.2

If there exists a state vector that satisfies a system predicate then for every interval of the system lifetime there exists a history such that the system predicate is satisfied at that time point.

More precisely,

$$\exists V \in \Gamma(\text{Sy}): V \text{ sat } \text{SysPred} \Rightarrow \forall Int \in \text{SI}(T): \exists H_{Int} \in \Gamma H: H_{Int} \text{ sat } \text{SysPred}@Int.$$

Proof.

This result follows from the fact that the universal history set is the cross product of Γ and T .

Lemma 4.3

For any three histories (say, H_w , H_x and H_y) and time points t_0 and t_1 ($t_0 < t_1$) there is another history (say, H_z) which is equivalent to H_w up to t_0 and equivalent to H_x after t_0 and up to t_1 and H_y after t_1 .

More precisely, $\forall H_w, H_x, H_y \in \Gamma H: \forall t_0, t_1 \in T: t_0 < t_1:$

$\exists H_z \in \Gamma H:$

$$(H_z|[s(T), t_0] = H_w|[s(T), t_0]) \wedge (H_z|(t_0, t_1) = H_x|(t_0, t_1)) \wedge (H_z|[t_1, e(T)] = H_y|[t_1, e(T)]).$$

Proof.

This result follows from the fact that the history H_z can be defined as follows:

$$H_z = H_w \text{ iff } t \in [s(T), t_0]$$

$$H_x \text{ iff } t \in (t_0, t_1)$$

$$H_y \text{ iff } t \in [t_1, e(T)].$$

Next we prove *theorem 4.1*, using the lemmas given above.

We have $\exists V \in \Gamma(\text{Sy}): V \text{ sat Start}(m) \wedge \text{Inv}(m) \wedge \neg \text{End}(m) \wedge C(\text{IR}(D)).$

\therefore (lemma 4.2) $\forall \text{Int} \in \text{SI}(T): \exists H_{\text{int}} \in \Gamma H:$

$$H_{\text{Int}} \text{ sat (Start}(m) \wedge \text{Inv}(m) \wedge \neg \text{End}(m) \wedge C(\text{IR}(D)))@ \text{Int}.$$

Similarly $\forall \text{Int} \in \text{SI}(T): \exists H_{\text{Int}} \in \Gamma H: H_{\text{Int}} \text{ sat (Inv}(m) \wedge \text{End}(m) \wedge C(\text{IR}(D)))@ \text{Int}.$

Choose any t and $\Delta t (> 0)$ such that it satisfies the lifetime constraint of the mode:

$$\text{LB}(m) \leq \Delta t \leq \text{UB}(m).$$

We have $\exists H_w \in \Gamma H: H_w \text{ sat (Start}(m) \wedge \text{Inv}(m) \wedge \neg \text{End}(m) \wedge C(\text{IR}(D)))@ t$

$$\exists H_x \in \Gamma H: H_x \text{ sat (Inv}(m) \wedge \neg \text{End}(m) \wedge C(\text{IR}(D)))@ [t, t + \Delta t]$$

and $\exists H_y \in \Gamma H: H_y \text{ sat (Inv}(m) \wedge \text{End}(m) \wedge C(\text{IR}(D)))@ t + \Delta t.$

\therefore (lemma 4.3) $\exists H_z \in \Gamma H:$

$$H_z \text{ sat (Start}(m) \wedge C(\text{IR}(D)))@ t \wedge$$

$$H_z \text{ sat (Inv}(m) \wedge C(\text{IR}(D)))@ [t, t + \Delta t] \wedge$$

$$H_z \text{ sat End}(m) \odot [t, t + \Delta t] \wedge$$

$$\text{LB}(m) \leq \Delta t \leq \text{UB}(m).$$

$\therefore \exists H \in \Gamma H: H \text{ sat } m@ [t, t + \Delta t] \wedge C(\text{IR}(D))@ [t, t + \Delta t].$

$\therefore \exists H \in \text{Iset}(D): H \text{ sat } m@ [t, t + \Delta t].$

$\therefore m \text{ con } D.$

Remark: In the above proof and later in theorems 4.2 and 4.3, we assume that the systems lifetime is such that it does not impose any restrictions on the satisfaction of the time bound constraints of a mode.

4.1.6. Mode Limitations

In this section I will point out some limitations (in expressive power) of a mode, by presenting a set of informal requirements, that describe the behaviour of a task, which cannot be expressed by a mode. The consequences of these limitations will then be discussed.

Consider the following informal requirements:

Reaction task

“The purpose of the reaction task is to initiate a reaction in the vessel and then to empty some of the contents into a different vessel; and to lower the temperature of the vessel. System behaviour during the task is defined informally by the following conditions.

- i) At the start of the task the level of liquid in the vessel (p_x) must be v_1 ($v_1 > 0$) and the temperature of the liquid (p_y) must be $temp_1$.*
- ii) The level of liquid in the vessel must remain at v_1 until the temperature of the liquid is raised to $temp_2$ (the activation temperature).*
- iii) The reaction must be initiated by raising the temperature of the vessel to $temp_2$.*
- iv) The task is completed when the volume of liquid in the vessel is at v_2 and the temperature is lowered to $temp_1$.”*

The following system predicates can be extracted from the requirements:

$$p_x = v_1, p_y = temp_1, p_x = v_1 \vee p_y = temp_2, p_x = v_2.$$

As a first attempt to specify the informal requirements of a mode we could construct the following mode:

$$\text{Reaction} = \langle p_x = v_1 \wedge p_y = temp_1, p_x = v_1 \vee p_y = temp_2, p_y = temp_1 \wedge p_x = v_2 \rangle.$$

To check the mode specification it is compared with the four conditions stipulated by the informal requirements of a task.

Condition i). It should be obvious that the start predicate of the mode ensures that *if a history satisfies the mode this condition is satisfied at the start of the mode.*

Condition ii). The start predicate and the invariant predicate ensure that *if a history satisfies the reaction mode then this condition is satisfied during the satisfaction of the mode.*

Condition iii). The invariant and end predicate ensure that *if a history satisfies the reaction*

mode then this condition is satisfied at some point during the satisfaction of the mode (if we assume that the variables p_x and p_y are continuous variables).

Condition iv). It should be obvious that the start predicate of the mode ensures that *if a history satisfies* the mode this condition is satisfied at the end of the mode.

From the above discussion we can conclude that *if a history satisfies* the reaction mode then during the satisfaction of the mode the informal conditions of the reaction task are satisfied.

However, a major flaw with the specification is realised when we check the consistency of the mode – invariant predicate and end predicate are in conflict. Hence the mode as defined above is inconsistent. (It will be shown in a later section that the informal requirements are indeed consistent.)

In fact no consistent mode can specify the informal requirements of the reaction task. The cause of the conflict is the fact that the behaviour of the task changes by the occurrence of an event (characterized by the predicate, $p_y = \text{temp}_2$) which marks the start of the reaction of the vessel. The key observation being that to ensure that the volume of liquid is at v_1 until a reaction is initiated the mode must impose a constraint over the temperature and volume during the lifetime of the mode, however a constraint is required only up to the time that a reaction is initiated. This example highlights the main restriction that should be imposed on the behaviour that can be expressed by a mode. Specifically, it shows that if the behaviour during a task is affected by events, then a mode should not be used to represent the behaviour of the task. In section 4.2.4 we show how this drawback can be overcome by introducing mode sequences.

4.1.7. Mode Benefits

The major benefit of using modes as the basis of a formal specification of a system is that they can be used to concisely represent the behaviour of tasks. However, as was pointed out in the previous section some tasks cannot be represented by modes. The class of tasks for which the construct of a mode is most appropriate are those tasks, for which the only events of interest are the events that mark initiation and the completion of the task.

The concept of a mode can also be used to structure the specification of a complex task

into several modes – by an identification of simple tasks. In such an approach all events of interest during a complex task would be determined, these events would then be used to partition the behaviour of the tasks into simple tasks – during which there is only one event of interest. The simple tasks of a complex task could then be specified using modes.

4.2. Mode Sequences

In this section I will introduce a formal construct, based on the concept of a mode, that can be used to specify the behaviour of a system for an interval, during which the behaviour is affected by (pre-defined) real-time events. The construct is a sequence of modes, called a mode sequence.

Definition: Mode sequence

A mode sequence is a finite sequence of modes, $ModeSeq = \langle m_1, \dots, m_r \rangle$, where m_i is a mode, for $i \in \{1, \dots, r\}$.

Semantics

The semantics of a mode sequence will be defined in terms of the satisfaction condition. We will say that a history satisfies a mode sequence of length r for an interval Int if and only if all there is a sequence of time points t_0, \dots, t_r such that m_i is satisfied during the closed interval given by the $i-1^{th}$ and i^{th} time points, and t_0 is the start point of the interval and t_r is the end point of the interval. (In other words, the modes in the mode sequence are satisfied by the history during the interval and in the order indicated by the mode sequence.)

More precisely, a history H satisfies a mode sequence $\langle m_1, \dots, m_r \rangle$ during an interval Int iff $\exists t_0, \dots, t_r: t_0 \leq \dots \leq t_r \wedge t_0 = s(Int) \wedge t_r = e(Int) \wedge H \text{ sat } m_i@[t_{i-1}, t_i]$, for $i = 1, \dots, r$.

We will denote this satisfaction as: $H \text{ sat } ModeSeq@Int$ (or $H \text{ sat } ModeSeq@ \langle t_0, \dots, t_r \rangle$)

In the special case where $Int = T$ we simply say that a history satisfies a mode sequence, which we denote by writing $H \text{ sat } ModeSeq$. When we wish to emphasize the difference between this special case and the general case, the special case is referred to as a lifetime mode sequence and the general case as an interval mode sequence.

Notation

A mode sequence will be denoted as ms ; if more than one mode sequence is used in an

expression, they will be denoted by ms_1, ms_2, \dots . The length of a mode sequence is simply the number of modes in the sequence; this is denoted by $|ms|$. The i th mode of a mode sequence ms will be denoted by $ms(i)$. The sequence of modes obtained by taking the i^{th} , $i + 1^{th}$ to j^{th} modes will be denoted by $ms(i, j)$ (i.e, $ms(i, j) = \langle ms(i), ms(i + 1), \dots, ms(j) \rangle$). The first mode of a sequence, $ms(1)$, will be referred to as the start mode and will be denoted by $S(ms)$; $ms(|ms|)$ will be referred to as the end mode and will be denoted by $E(ms)$. To simplify expressions over the components of a mode the following abbreviations will be used: $Start_i$, Inv_i and End_i for $Start(ms(i))$, $Inv(ms(i))$ and $End(ms(i))$ respectively.

4.2.1. Mode Sequence Properties

In this section I will discuss some properties of mode sequences which can be derived from the definition of a mode sequence, and the properties of a mode.

Lemma 4.4.

If a history H of a system satisfies a mode sequence ms for an interval then there is a unique sequence of time points $\langle t_0, \dots, t_{|ms|} \rangle$ in the interval which specify the start and end points of the modes in the mode sequence.

More precisely,

$$\forall H \in \Gamma H: [\quad H \text{ sat } ms@Int \Rightarrow$$

$$\exists ! t_0 \leq \dots \leq t_{|ms|}: t_0 = s(Int) \wedge t_{|ms|} = e(Int) \wedge \forall i \in \{1, \dots, |ms|\}: H \text{ sat } ms(i)@[t_{i-1}, t_i] \quad].$$

Proof. The fact that a sequence of time points exists is obvious from the semantics of a mode sequence. Such a sequence is unique since for each interval $[t_{i-1}, t_i]$, $i \in \{1, \dots, |ms|\}$, t_i is the earliest time point at which the end predicate of $ms(i)$ holds after t_{i-1} .

Lemma 4.5.

If a history H of a system satisfies a mode sequence ms for the sequence of $|ms| + 1$ time points $\langle t_0, \dots, t_{|ms|} \rangle$ then the invariant predicate and end predicate of $ms(i)$ and the start predicate and invariant predicate of $ms(i + 1)$ must be satisfied at time point t_i .

More precisely,

$$\forall H \in \Gamma H: [\quad H \text{ sat } ms@\langle t_0, \dots, t_{|ms|} \rangle$$

$$\Rightarrow \forall i \in \{1, \dots, |ms| - 1\}: H \text{ sat } (Inv_i \wedge End_i \wedge Start_{i+1} \wedge Inv_{i+1})@t_i \quad].$$

Proof. This result follows directly from the semantics of a mode sequence.

Mode Sequence Residence

To capture formally what is meant by when we say a history resides in a mode sequence during an interval, the notion of mode residence is introduced.

Definition: Mode sequence residence

A system history resides in a mode sequence during a specified interval if and only if the interval is a sub interval of an interval in which the history satisfies the mode sequence.

More precisely, $H \text{ res } ms@Int_0$ iff $(\exists Int_1 \in SI(T): H \text{ sat } ms@Int_1 \wedge Int_0 \subseteq Int_1)$.

Sequence Components

For a mode sequence we will define a start predicate, invariant predicate and end predicate. The start predicate is defined as the start predicate of the start mode, the invariant predicate as the disjunction of all the invariant predicates of the modes of the sequence and the end predicate will be defined as the end predicate of the end mode. For a mode sequence (ms) the start predicate will be denoted by $SeqStart(ms)$, the invariant predicate by $SeqInv(ms)$ and the end predicate by $SeqEnd(ms)$. When the mode sequence is obvious from the context the following abbreviations will be used, $SeqStart$, $SeqInv$ and $SeqEnd$. The formal definitions of the components are:

$SeqStart =_{\text{def}} Start(S(ms)); SeqInv =_{\text{def}} Inv_1 \vee \dots \vee Inv_{|ms|}$ and $SeqEnd =_{\text{def}} End(E(ms))$.

Lemma 4.6.

If a history satisfies a mode sequence during an interval, then the history satisfies the mode sequence invariant during the interval.

$\forall H \in \Gamma H: \forall Int \in SI(T): H \text{ sat } ms@Int \Rightarrow H \text{ sat } SeqInv(ms)@Int$.

Proof.

This result follows from the semantics of mode sequence satisfaction and the definition of a sequence invariant.

Satisfaction Set

The behaviours of a system that are specified by a lifetime mode sequence for a given history description can be captured by the concept of a satisfaction set, which is the set of all possible behaviours from a history description which satisfy the mode sequence.

Definition: Satisfaction set

The satisfaction set of a mode sequence ms for a history set HH is the set of histories in HH which satisfy the mode sequence during the system lifetime. This is denoted by $Hset(HH, ms)$.

More precisely, $Hset(HH, ms) = \{H \in HH: H \text{ sat } ms\}$.

4.2.2. Mode Sequence Relationships

Equivalence and implication relations can be defined over a set of mode sequences, in the same way as was done for modes.

Definition: Mode sequence equivalence

Two mode sequences are equivalent for a set of histories HH if and only if all histories of the set which satisfy one mode sequence during an interval satisfy the other mode sequence during the interval.

More precisely, $ms_1 \simeq^{HH} ms_2$ iff $\forall H \in HH: \forall Int \in SI(T): [H \text{ sat } ms_1 @ Int \Leftrightarrow H \text{ sat } ms_2 @ Int]$.

Definition: Mode sequence implication.

A mode sequence ms_1 implies another mode sequence ms_2 for a set of histories, HH if and only if all histories in the set which satisfy ms_1 during an interval satisfy ms_2 during that interval (in which case we say that ms_2 is a consequent of ms_1).

More precisely, $ms_1 \sim^{HH} ms_2$ iff $\forall H \in HH: \forall Int \in SI(T): [H \text{ sat } ms_1 @ Int \Rightarrow H \text{ sat } ms_2 @ Int]$.

4.2.3. Mode Sequence Consistency

The notion of consistency introduced for modes in section 4.1.5 can be extended to modes sequences. The satisfaction condition of a modes sequence can be used to define a consistent mode sequence for a given history description set, in the same way as was done for modes.

Definition: Mode Sequence Consistency

A mode sequence ms is consistent, for a set of invariant histories of a description D , if it is possible for at least one history of the set to satisfy the mode sequence. This will be denoted by: $ms \text{ con } D$.

More precisely, $ms \text{ con } D$ iff: $\exists H \in Iset(D): \exists Int \in SI(T): H \text{ sat } ms @ Int$.

Mode Sequence Consistency Checks

The consistency of a mode sequence is checked by confirming the following three conditions.

i) *There exists a state value that satisfies the start predicate, invariant predicate, negation of the end predicate of the start mode of the mode sequence and the invariant relations.*

More precisely, $\exists V \in \Gamma: V \text{ sat Start}(S(ms)) \wedge \text{Inv}(S(ms)) \wedge \neg \text{End}(S(ms)) \wedge C(IR(D))$.

ii) *For the i th mode (for $i = 2, \dots, |ms|$) of the mode sequence there exists a state value that satisfies the conjunction of the invariant and end predicate of the $i-1$ th mode and the start predicate, invariant predicate and negation of the end predicate of the i th mode and the invariant relations.*

More precisely,

$\forall i \in \{2, \dots, |ms|\}: \exists V \in \Gamma: [V \text{ sat Inv}(ms(i-1)) \wedge \text{End}(ms(i-1)) \wedge \text{Start}(ms(i)) \wedge \text{Inv}(ms(i)) \wedge \neg \text{End}(ms(i)) \wedge C(IR(D))]$.

iii) *There exists a state value that satisfies the conjunction of the invariant and end predicate of the end mode of the mode sequence and the invariant relations.*

More precisely, $\exists V \in \Gamma: [V \text{ sat Inv}(E(ms)) \wedge \text{End}(E(ms)) \wedge C(IR(D))]$.

Mode Sequence Consistency Theorem

The mode sequence consistency theorem shows that if the mode sequence consistency checks (given above) are confirmed for a mode sequence, against a given history description, then that mode is consistent for the history description.

Theorem 4.2

If the mode consistency checks hold for a description D then the mode sequence is consistent for that history description.

More precisely,

$(\exists V \in \Gamma(Sy): V \text{ sat Start}(S(ms)) \wedge \text{Inv}(S(ms)) \wedge \neg \text{End}(S(ms)) \wedge C(IR(D))) \wedge$
 $(\forall i \in \{2, \dots, |ms|\}: \exists V \in \Gamma: [V \text{ sat Inv}(ms(i-1)) \wedge \text{End}(ms(i-1)) \wedge \text{Start}(ms(i)) \wedge \text{Inv}(ms(i)) \wedge \neg \text{End}(ms(i)) \wedge C(IR(D))]) \wedge$
 $(\exists V \in \Gamma(Sy): [V \text{ sat Inv}(E(ms)) \wedge \text{End}(E(ms)) \wedge C(IR(D))])$
 $\Rightarrow ms \text{ con } D.$

Proof.

We have $\exists V \in \Gamma(\text{Sy}): V \text{ sat } \text{Start}(S(\text{ms})) \wedge \text{Inv}(S(\text{ms})) \wedge \neg \text{End}(S(\text{ms})) \wedge C(\text{IR}(\text{D}))$.

\therefore (lemma 4.2) $\forall \text{Int} \in \text{SI}(\text{T}): \exists H_{\text{int}} \in \Gamma H:$

$$H_{\text{Int}} \text{ sat } (\text{Start}(S(\text{ms})) \wedge \text{Inv}(S(\text{ms})) \wedge \neg \text{End}(S(\text{ms})) \wedge C(\text{IR}(\text{D})))@_{\text{Int}}$$

Similarly $\forall i \in \{2, \dots, |\text{ms}|\}: \forall \text{Int} \in \text{SI}(\text{T}): \exists H_{\text{Int}} \in \Gamma H:$

$$H_{\text{Int}} \text{ sat } (\text{Inv}(\text{ms}(i-1)) \wedge \text{End}(\text{ms}(i-1)) \wedge \text{Start}(\text{ms}(i)) \wedge \text{Inv}(\text{ms}(i)) \wedge \neg \text{End}(\text{ms}(i)) \wedge C(\text{IR}(\text{D})))@_{\text{Int}}.$$

Also $\forall \text{Int} \in \text{SI}(\text{T}): \exists H_{\text{int}} \in \Gamma H:$

$$H_{\text{Int}} \text{ sat } (\text{Inv}(E(\text{ms})) \wedge \text{End}(E(\text{ms})) \wedge C(\text{IR}(\text{D})))@_{\text{Int}}.$$

Choose $t_0, \dots, t_{|\text{ms}|}$ such that the lifetime constraints of the modes are satisfied

$$\text{LB}(\text{ms}(i)) \leq (t_i - t_{i-1}) \leq \text{UB}(\text{ms}(i)).$$

Suppose $H_x \text{ sat } (\text{Inv}(\text{ms}(i-1)) \wedge \text{End}(\text{ms}(i-1)) \wedge$

$$\text{Start}(\text{ms}(i)) \wedge \text{Inv}(\text{ms}(i)) \wedge \neg \text{End}(\text{ms}(i)) \wedge C(\text{IR}(\text{D})))@_{t_{i-1}}$$

$$H_y \text{ sat } (\text{Inv}(\text{ms}(i-1)) \wedge \text{End}(\text{ms}(i-1)))@_{[t_{i-1}, t_i]}$$

and $H_z \text{ sat } (\text{Inv}(\text{ms}(i)) \wedge \text{End}(\text{ms}(i)) \wedge$

$$\text{Start}(\text{ms}(i+1)) \wedge \text{Inv}(\text{ms}(i+1)) \wedge \neg \text{End}(\text{ms}(i+1)) \wedge C(\text{IR}(\text{D})))@_{t_i}$$

\therefore (lemma 4.3) $\exists H_i \in \Gamma H:$

$$H_i \text{ sat } (\text{Start}(\text{ms}(i)) \wedge \text{Inv}(\text{ms}(i-1)) \wedge \text{End}(\text{ms}(i-1)) \wedge C(\text{IR}(\text{D})))@_{t_{i-1}} \wedge$$

$$H_i \text{ sat } (\text{Inv}(\text{ms}(i)) \wedge C(\text{IR}(\text{D})))@_{[t_{i-1}, t_i]} \wedge$$

$$H_i \text{ sat } \odot \text{End}(\text{ms}(i))@_{[t_{i-1}, t_i]} \wedge$$

$$H_i \text{ sat } \text{Start}(\text{ms}(i+1)) \wedge \text{Inv}(\text{ms}(i+1))@_{t_i} \wedge$$

$$\text{LB}(\text{ms}(i)) \leq (t_i - t_{i-1}) \leq \text{UB}(\text{ms}(i)).$$

$\therefore \exists H_i \in \Gamma H: H_i \text{ sat } \text{ms}(i)@_{[t_{i-1}, t_i]} \wedge$

$$H_i \text{ sat } (C(\text{IR}(\text{D})))@_{[t_{i-1}, t_i]} \wedge$$

$$H_i \text{ sat } \text{Inv}(\text{ms}(i-1)) \wedge \text{End}(\text{ms}(i-1))@_{t_{i-1}} \wedge$$

$$H_i \text{ sat } \text{Start}(\text{ms}(i+1)) \wedge \text{Inv}(\text{ms}(i+1))@_{t_i}.$$

Define a history H as follows:

$$H(t) = H_i(t) \text{ iff } t \in [t_{i-1}, t_i], \text{ for } i = 2, \dots, |\text{ms}|-1,$$

$$= H_1(t) \text{ iff } t \in [s(T), t_1)$$

$$= H_{|ms|}(t) \text{ iff } t \in [t_{|ms|-1}, e(T)].$$

$$\therefore \exists H \in \Gamma H: H \text{ sat } ms@[t_0, t_{|ms|}] \wedge$$

$$H \text{ sat } (C(IR(D)))@[t_0, t_{|ms|}]$$

$$\therefore \exists H \in Iset(D): H \text{ sat } ms@[t_0, t_{|ms|}]$$

$$\therefore ms \text{ con } D.$$

4.2.4. Mode Sequence Example

In this section I will discuss (by example) how informal requirements over the system can be expressed concisely as a mode sequence. The example given is that of the reaction task (introduced in section 4.1.6).

Reaction task

“The purpose of the reaction task is to initiate a reaction in the vessel and then to empty some of the contents into a different vessel; and the lower the temperature of the vessel. System behaviour during the task is defined informally by the following conditions.

- i) At the start of the task the level of liquid in the vessel (p_x) must be v_1 ($v_1 > 0$) and the temperature of the liquid (p_y) must be $temp_1$.*
- ii) The level of liquid in the vessel must remain at v_1 until the temperature of the liquid is raised to $temp_2$ (the activation temperature).*
- iii) The reaction must be initiated by raising the temperature of the vessel to $temp_2$.*
- iv) The task is completed when the volume of liquid in the vessel is at v_2 and the temperature is lowered to $temp_1$.”.*

The following system predicates can be extracted from the requirements:

$$p_x = v_1, p_y = temp_1, p_y = temp_2 \text{ and } p_x = v_2.$$

There are two events of interest during the reaction task:

$$p_y = temp_2 \text{ and } p_x = v_2 \wedge p_y = temp_1.$$

These events can be used to partition the reaction task into two simpler tasks: *Activate* and *Empty*.

Reaction = $\langle \text{Activate}, \text{Empty} \rangle$,

where Activate = $\langle p_x = v_1 \wedge p_y = \text{temp}_1, p_x = v_1, p_y = \text{temp}_2 \rangle$,

 Empty = $\langle \text{true}, \text{true}, p_x = v_2 \wedge p_y = \text{temp}_1 \rangle$.

To check the mode sequence it is compared with the four conditions stipulated by the informal requirements of a task.

Condition i). The start predicate of the Activate mode ensures that *if a history satisfies* the mode sequence this condition is satisfied at the start of the mode.

Condition ii). The invariant predicate and the end predicate of the Activate mode ensure that *if a history satisfies* the mode sequence then this condition is satisfied during the satisfaction of the mode.

Condition iii). The end predicate of the activate mode ensures that *if a history satisfies* the mode sequence then this condition is satisfied at some point during the satisfaction of the mode sequence.

Condition iv). The end predicate of the empty mode ensures that *if a history satisfies* the mode sequence this condition is satisfied at the end of the mode.

It should be obvious that both modes will be consistent, for some history description.

Comments

Recall that the reaction task could not be specified, because the event which marks the start of the reaction (we will refer to this as the *reaction event*) influences the behaviour of the task. The mode sequence given above was able to overcome the problem caused by the *reaction event* by splicing the task into two simpler tasks. By splitting the task the invariant over the volume of liquid in the vessel can be imposed up to the start of the reaction, without having to impose it during all of the reaction task.

4.2.5. Mode Sequence Set

The specifications produced during the requirements analysis may be expressed as a set of mode sequences. The mode sequences of the set must be imposed over a common history description.

Definition: Mode sequence set

A mode sequence set (MSS) is a set of mode sequences, in which all the mode sequences are

imposed over a common history description.

Semantics

We will say that a history H satisfies a mode sequence set MSS (denoted by $H \text{ sat } MSS$) if and only if it satisfies a mode sequence of the set (i.e., $H \text{ sat } MSS \Leftrightarrow \exists ms \in MSS: H \text{ sat } ms$).

Equivalence and implication relations can be defined over a set of mode sequence sets, in the same way as was done for mode sequences. Formally, these relations are defined as:

$MSS_1 \approx^{HH} MSS_2$ iff $\forall H \in HH: [H \text{ sat } MSS_1 \Leftrightarrow H \text{ sat } MSS_2]$; and

$MSS_1 \sim^{HH} MSS_2$ iff $\forall H \in HH: [H \text{ sat } MSS_1 \Rightarrow H \text{ sat } MSS_2]$.

4.2.6. Mode Sequence Limitations

We normally start from an intuitive informal requirements, and need to build a mode sequence set MSS to express these. Then we need to check MSS against our intuition.

Three drawbacks of representing a requirements specification as a set of mode sequences are pointed out below.

i) Concise specifications

A mode sequence, can only represent a linear sequence of tasks. Each possible sequence of modes (tasks) that reflects the informal requirements, must be represented by a mode sequence in the mode set. Consider a system with many possible mode sequences, the mode sequence set specification of such a system would be large and unstructured. Such a representation would be difficult to develop and check, since much of the information would be repeated and there would be no overall structure to the specification.

ii) System Structure

A mode sequence set does not clarify the relationship between the modes. For example if we wish to check the modes that can precede a specific mode, all mode sequences of the set would have to be checked. The major reason why the relationship is not explicit, is that the development and representation of a specification in terms of mode sequences leads to a linear perspective over the system.

A major consequence of the absence of an explicit relationship between the modes is

that the structure of the system, in terms of the events, is not made apparent during the development of the specification.

iii) Consistency and Completeness

Checks for consistency are complicated since the consistency of each mode sequence would have to be checked. There seems no obvious intuitive definition of completeness for a mode sequence set (a formal definition of completeness for an alternative representation will be given later).

Alternative Representation

It is a generally acknowledged fact that the usability of a specification technique can be greatly enhanced if an intuitive feeling of the required behaviour can be deduced from a suitable presentation of the specification [Hat187, Pete84]. In the next section a graphical representation of a mode sequence set – *mode graph* – is proposed. The nodes of a mode graph will represent the modes and the arcs will represent the transitions between the modes. In the following paragraphs, I will point out how mode graphs can help to overcome the drawbacks of a mode sequence set are discussed.

i) Concise representation.

Mode graphs are a more concise representation of a mode sequence set (as opposed to a set of sequences), since a mode graph will have only one node for each mode. Consider a system with many possible mode sequences, but a few modes, the formal mode graph representation of the requirements specification would be concise – in the sense that only the necessary information would be represented. Furthermore the picture of the specification makes it easier to obtain a feel for the behaviour of the system.

ii) System structure.

The relationship between the modes is explicitly represented by a mode graph. For example, the modes which may follow a given are apparent from the structure of the graph. The structure of the graph reflects the structure of the system as formed by the events (that mark the completion of tasks) and makes it possible to identify and represent closely related events (i.e., events associated with a high-level task). A further benefit is gained

during the development of the specification – since to the development of a mode graph leads to a more structured approach to the analysis of the informal requirements.

iii) Consistency and Completeness

As will be shown in the next section it is possible to develop simple checks for the consistency and completeness of specifications represented by mode graphs.

4.3. Mode Graphs

In this section I introduce a (formal) graphical representation of a requirements specification for which the semantics are based on the concept of a mode sequence set. Roughly speaking, the formal graph is specified over a unified set of modes, in which the modes are represented by nodes and the arcs represent the transitions between modes. These graphs will be referred to as mode graphs. A formal definition of a mode graph is given below.

Definition: Mode graph

A mode graph (MG) is a four-tuple $MG = \langle M, A, S, E \rangle$, where M is a (finite) set of unified modes, A is a set of mode pairs (i.e., $A \subseteq M \times M$), S and E are non-empty subsets of M (i.e., $S, E \subseteq M, S, E \neq \emptyset$). The members of set S are referred to as start modes and the members of set E as end modes. (Thus, MG can be viewed as a digraph with nodes M and edges E).

For a mode graph MG the following conditions must hold.

(To simplify the conditions it will be assumed that the mode set (M) is a set of indexed modes represented as: $\{m_1, \dots, m_{|M(MG)|}\}$.)

i) A must be *irreflexive*, i.e., $\forall (m_i, m_j) \in A: i \neq j$.

In the definition of the following two conditions we will make use of the reflexive transitive closure set ($RTC(MG)$) of the mode graph; roughly speaking $(m_i, m_j) \in RTC(MG)$ if and only if there exists a path from m_i to m_j in MG . More precisely, $(x, y) \in RTC(MG)$ iff there exist modes w_0, \dots, w_s such that $x = w_0 A w_1 A w_2 \dots w_{s-1} A w_s = y$. The reflexive transitive closure of a mode graph can be constructed by using Warshall's algorithm [Gers87].

ii) For any mode there is a *path to it from a start mode*,

i.e., $\forall m_j \in M: \exists m_i \in S: (m_i, m_j) \in RTC$.

iii) For any mode there is a *path from it to an end mode*,

i.e., $\forall m_i \in M: \exists m_j \in E: (m_i, m_j) \in RTC$.

Semantics

To define the semantics of a mode graph, a precise definition of the set of mode sequences specified by a mode graph is required. These mode sequences are specified by paths in the mode graph which start at a node labelled by a start mode and end at a node labelled by an end mode. For a mode graph (MG) this set will be referred to as the graph sequence set and denoted by $Seq(MG)$. More precisely,

$Seq(MG) =$

$\{ms \in M(MG)^+ \mid$

$(ms(i), ms(i+1)) \in A(MG) \wedge ms(1) \in S(MG) \wedge ms(|ms|) \in E(MG), \text{ for } i = 1, \dots, |ms|-1\}$

We will say that a history H of a history description satisfies a mode graph MG during an interval Int if and only if the history satisfies a mode sequence in the graph sequence set, of MG , during Int . That is, a history H satisfies a mode graph MG during interval Int if and only if: $\exists ms \in Seq(MG): H \text{ sat } ms@Int$; this satisfaction is denoted in the normal way by writing $H \text{ sat } MG@Int$. Similarly we can define mode graph satisfaction for a history H during the system lifetime as: $H \text{ sat } MG$ iff $\exists ms \in Seq(MG): H \text{ sat } ms$.

Notation

The mode set, set of arcs, start set and end set of a mode graph MG will be denoted by $M(MG)$, $A(MG)$, $S(MG)$ and $E(MG)$ respectively.

Pictorial Representation

A picture of the mode graph $MG = \langle M, A, S, E \rangle$ is a diagram of nodes (annotated ellipses) corresponding to the members of M , and arrows corresponding to the members of A , such that if (m_i, m_j) is a member of A then there is an arrow which goes from the node labelled by m_i to the node labelled by m_j . To emphasize the start and end modes, the start modes will have a broken boundary and the end modes will be shaded.

From the conditions on the structure of a mode graph we can make the following two observations of the picture: i) there are no self loops in a mode graph (from condition i);

and ii) there are no multiple arcs in the mode graph (from the representation of the arcs).

In figure 4.3, three graphs are drawn and the (mode graph) structural conditions are used to check if the graphs could be mode graphs. The first graph (graph a) is not a well-defined mode graph since the graph does not satisfy conditions ii) and iii). The second (graph b) and third (graph c) graphs are well-defined mode graphs. The second graph is an example of an unconnected mode graph; whereas the third is a connected mode graph.

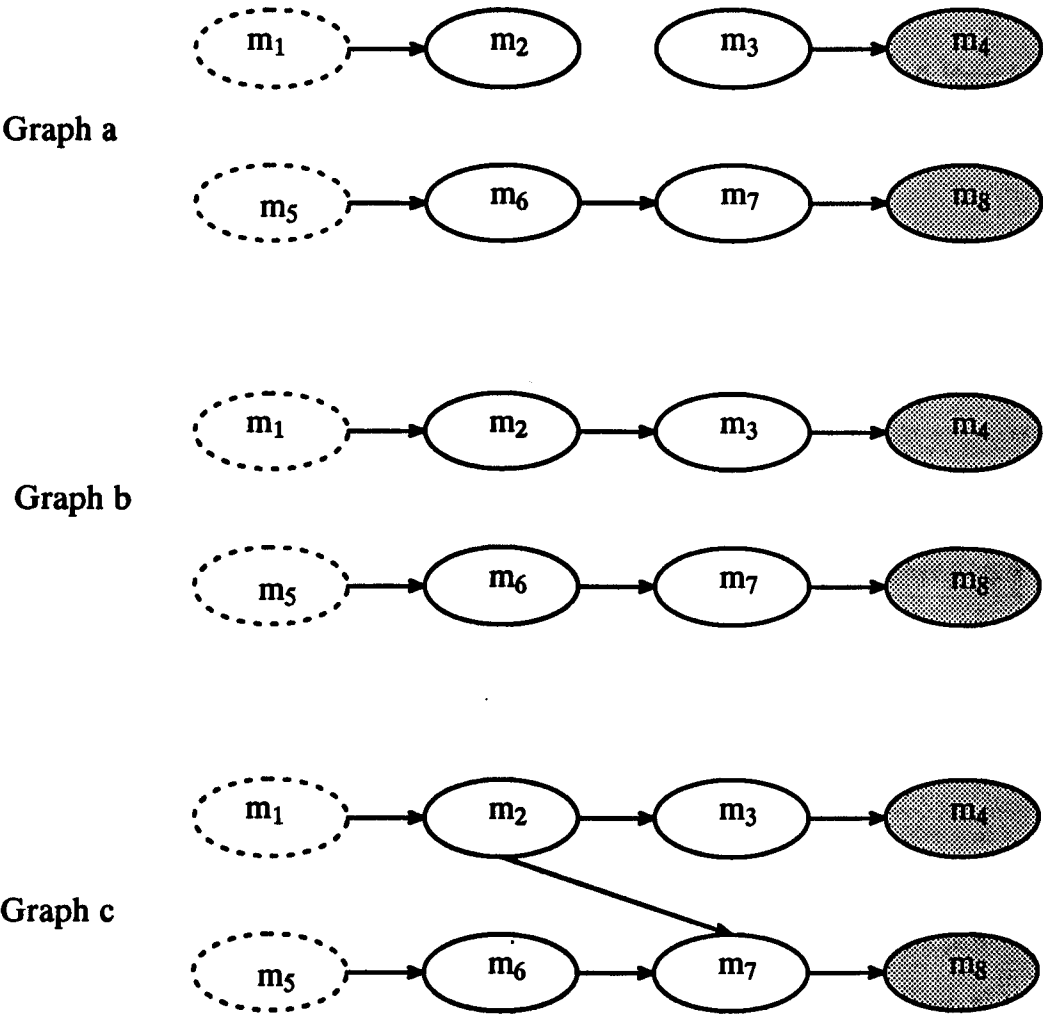


Figure 4.3. Mode Graphs

4.3.1. Mode Graph Properties

In this section, I will discuss some properties which follow from the definition of the semantics of a mode graph.

Lemma 4.7

If a history satisfies a mode graph during any interval then there is a mode sequence in the graph sequence set, and a unique sequence of time points at which that mode sequence is satisfied.

More precisely,

$\forall H \in \Gamma H: \forall \text{Int} \in \text{SI}(T): H \text{ sat } MG @ \text{Int} \Rightarrow$

$\exists ms \in \text{Seq}(MG): \exists! t_0, \dots, t_{|ms|}: H \text{ sat } ms @ \langle t_0, \dots, t_{|ms|} \rangle \wedge t_0 = s(\text{Int}) \wedge t_{|ms|} = e(\text{Int}).$

Proof. This result follows from lemma 4.3 and the satisfaction condition of a mode graph.

Satisfaction Set

The concept of the satisfaction set of a mode sequence (i.e., the set of histories satisfying the mode sequence) can be extended to a mode graph, as the set of all possible behaviours from a history description which satisfy the mode graph.

Definition: Satisfaction Set

For a given set of histories HH we can define the satisfaction set of a mode graph MG as the set of histories which satisfy the mode graph. This is denoted by Hset(HH, MG).

More precisely, $\text{Hset}(HH, MG) =^{\text{def}} \{H \in HH: H \text{ sat } MG\}.$

4.3.2. Mode Graph Components

For a mode graph we can define a start predicate, an invariant predicate and an end predicate. The start predicate is defined as the disjunction of the start predicates of the modes in the start set, the invariant predicate is defined as the disjunction of the invariant predicates of the modes in the mode set and the end predicate is defined as the disjunction of all the end predicates of the modes in the end set. For a mode graph (MG) the start predicate will be denoted by GStart(MG), the invariant predicate by GInv(MG) and the end predicate by GEnd(MG). When the graph is obvious from the context the following abbreviations will be used, GStart, GInv and GEnd. The formal definitions are:

$GStart(MG) = \bigvee_{m \in S(MG)} Start(m)$; $GInv(MG) = \bigvee_{m \in M(MG)} Inv(m)$ and $GEnd(MG) = \bigvee_{m \in E(MG)} End(m)$.

Lemma 4.7

If a history satisfies a mode graph then during the history the invariant predicate of the mode graph must hold (i.e., $\forall H \in \Gamma H: H \text{ sat } MG \Rightarrow H \text{ sat } GInv(MG)$).

Proof. Immediate from the semantics of a mode graph; and the definition of a graph invariant.

Graph Functions

In this section I will introduce some functions which can be used to reason about the relationship between modes as specified by a mode graph.

Predecessor Function

The predecessor function is a function which defines an *immediate* predecessor relationship between the modes of a graph. Roughly speaking, a mode is a predecessor of another mode in a mode graph if the mode can immediately precede the other mode in a mode sequence of the graph.

Definition: Predecessor function

The predecessor function of a mode graph MG (denoted by $MG.pr(m)$) is a function from a mode (say, m) of the mode graph to the set of all modes connected to m by an arc, in which the mode m is the second mode of the arc.

More precisely,

$MG.pr: M(MG) \rightarrow \text{Powerset}(M(MG)); MG.pr(m) = \{m' \in M(MG) \mid \exists (m', m) \in A(MG)\}$.

Successor Function

The successor function is a function which defines an *immediate* successor relationship between the modes of a graph. Roughly speaking, a mode is a successor of another mode in a mode graph if the mode can immediately succeed the other mode in a mode sequence of the graph.

Definition: Successor function

The successor function of a mode graph MG (denoted by $MG.sr(m)$) is a function from a mode (say, m) of the mode graph to the set of all modes connected to m by an arc, in which the mode m is the first mode of the arc.

More precisely,

$$MG.sr: M(MG) \rightarrow \text{Powerset}(M(MG)); MG.sr(m) = \{m' \in M(MG) \mid \exists(m, m') \in A(MG)\}.$$

4.3.3. Complete Mode Graphs

In this section a formal notion of what constitutes a complete specification (in terms of a mode graph) is presented. The notion of completeness, for a given set of histories HH , can be stated roughly as: if a mode of the mode graph is satisfied by a history from the set HH for an interval then a successor of the mode must be start satisfied at the end of the interval. Hence, the behaviour that must be exhibited by the system after the satisfaction of a mode has been *completely* specified (i.e., it has been specified for all system conditions that can exist at the end point of the satisfaction of a mode).

Definition: Mode graph completeness

We will say that a mode graph (MG) is complete for a history description D if and only if the satisfaction of any mode during any interval implies the satisfaction of the start predicate and invariant predicate of a mode in the successor set of the mode at the end of the interval, or the mode has no successors. This is denoted by $MG \text{ cmp } D$.

More precisely,

$MG \text{ cmp } D$ iff

$$\forall m \in M(MG): \forall H \in \text{Set}(D): \forall \text{Int} \in \text{SI}(T):$$

$$[H \text{ sat } m@Int \Rightarrow \exists x \in MG.sr(m): H \text{ sat } (\text{Start}(x) \wedge \text{Inv}(x))@e(Int)] \vee MG.sr(m) = \emptyset.$$

Remark. A mode m from $M(MG)$ that satisfies the following condition:

$$\forall H \in \text{Set}(D): \forall \text{Int} \in \text{SI}(T):$$

$$[H \text{ sat } m@Int \Rightarrow \exists x \in MG.sr(m): H \text{ sat } (\text{Start}(x) \wedge \text{Inv}(x))@e(Int)] \vee MG.sr(m) = \emptyset,$$

will be referred to as a complete mode.

4.3.4. Consistent Mode Graphs

The notion of consistency introduced for modes in section 4.1.5 can be extended to mode graphs. The satisfaction condition of a mode graph can be used to define a consistent mode graph for a given history description set, in the same way as was done for modes (and mode sequences).

Definition: Mode Graph Consistency

A mode graph MG is consistent, for a set of invariant histories of a description D, if all the mode sequences of the graph are consistent. This will be denoted by: MG con D.

More precisely, MG con D iff $\forall ms \in Seq(MG): ms \text{ con } D$.

Mode Graph Consistency Checks

The consistency of a mode graph is checked by confirming the following three conditions.

i) *For each start mode there exists a state value that satisfies the start predicate, invariant predicate, negation of the end predicate and the invariant relations.*

More precisely,

$\forall x \in S(MG): \exists V \in \Gamma: [V \text{ sat } Start(x) \wedge Inv(x) \wedge \neg End(x) \wedge C(IR(D))]$.

ii) *For each arc in the mode graph there exists a state value that satisfies the conjunction of the invariant and end predicate of the first mode and the start predicate, invariant predicate and negation of the end predicate of the second mode and the invariant relations.*

More precisely,

$\forall (x, y) \in A(MG): \exists V \in \Gamma:$

$[V \text{ sat } Inv(x) \wedge End(x) \wedge Start(y) \wedge Inv(y) \wedge \neg End(y) \wedge C(IR(D))]$.

iii) *For each end mode there exists a state value that satisfies the conjunction of the invariant, end predicate and the invariant relations.*

More precisely,

$\forall x \in E(MG): \exists V \in \Gamma(Sy): [V \text{ sat } Inv(x) \wedge End(x) \wedge C(IR(D))]$.

Remark. An arc (x, y) from $A(MG)$ that satisfies the following condition:

$\exists V \in \Gamma: [V \text{ sat } Inv(x) \wedge End(x) \wedge Start(y) \wedge Inv(y) \wedge \neg End(y) \wedge C(IR(D))]$,

will be referred to as a consistent arc.

Mode Graph Consistency Theorem

The mode graph consistency theorem shows that if the mode graph consistency check (given above) is confirmed for a mode graph, against a given history description, then that mode graph is consistent for that history description.

Theorem 4.3

If the mode graph consistency checks hold for a history description then the mode graph is consistent for that history description.

More precisely,

$$\begin{aligned}
 & (\forall x \in S(MG): \exists V \in \Gamma: [V \text{ sat } \text{Start}(x) \wedge \text{Inv}(x) \wedge \neg \text{End}(x) \wedge C(\text{IR}(D))]) \wedge \\
 & (\forall (x, y) \in A(MG): \exists V \in \Gamma: [V \text{ sat } \text{Inv}(x) \wedge \text{End}(x) \wedge \text{Start}(y) \wedge \text{Inv}(y) \wedge \neg \text{End}(y) \wedge C(\text{IR}(D))]) \wedge \\
 & (\forall x \in E(MG): \exists V \in \Gamma: [V \text{ sat } \text{Inv}(x) \wedge \text{End}(x) \wedge C(\text{IR}(D))]) \\
 & \Rightarrow \text{MG con } D.
 \end{aligned}$$

Proof.

A proof can be given by showing that the mode graph consistency condition ensures that the mode sequence conditions must hold for any mode sequence of MG.

Consider an arbitrary mode sequence ms of MG.

$$\begin{aligned}
 \text{We have} \quad & \forall x \in S(MG): \exists V \in \Gamma: [V \text{ sat } \text{Start}(x) \wedge \text{Inv}(x) \wedge \neg \text{End}(x) \wedge C(\text{IR}(D))] \\
 \therefore \quad & \exists V \in \Gamma: [V \text{ sat } \text{Start}(S(ms)) \wedge \text{Inv}(S(ms)) \wedge \neg \text{End}(S(ms)) \wedge C(\text{IR}(D))] \\
 \text{We have} \quad & \forall (x, y) \in A(MG): \exists V \in \Gamma: \\
 & V \text{ sat } \text{Inv}(x) \wedge \text{End}(x) \wedge \text{Start}(y) \wedge \text{Inv}(y) \wedge \neg \text{End}(y) \wedge C(\text{IR}(D))
 \end{aligned}$$

For all i , the pair $(ms(i-1), ms(i))$ must be in the arc set of the graph.

$$\therefore \quad \forall i \in \{2, \dots, |ms|\}: \exists V \in \Gamma: [V \text{ sat } \text{Inv}(ms(i-1)) \wedge \text{End}(ms(i-1)) \wedge \text{Start}(ms(i)) \wedge \text{Inv}(ms(i)) \wedge \neg \text{End}(ms(i)) \wedge C(\text{IR}(D))].$$

$$\begin{aligned}
 \text{We have} \quad & (\forall x \in E(MG): \exists V \in \Gamma: [V \text{ sat } \text{Inv}(x) \wedge \text{End}(x) \wedge C(\text{IR}(D))]). \\
 \therefore \quad & \exists V \in \Gamma: [V \text{ sat } \text{Inv}(E(ms)) \wedge \text{End}(E(ms)) \wedge C(\text{IR}(D))].
 \end{aligned}$$

$$\therefore (\text{theorem 4.2}) \quad ms \text{ con } D$$

$$\therefore \quad \forall ms \in \text{Seq}(MG): ms \text{ con } D$$

$$\therefore \quad \text{MG con } D.$$

4.3.5. Mode Graph Relationships

Equivalence and implication relations can be defined over mode graphs, in the same way as was done for modes and mode sequences. In addition, two relations over the structure of a graph will be defined: mode graph isomorphism and mode graph congruence.

Definition: Mode graph equivalence

Two mode graphs are equivalent (for a set of histories HH) if and only if all histories of the set which satisfy one mode graph satisfy the other mode graph.

More precisely, $MG_1 \simeq^{HH} MG_2$ iff $\forall H \in HH: H \text{ sat } MG_1 \Rightarrow H \text{ sat } MG_2$.

Definition: Mode graph implication.

A mode graph implies another mode graph (for a set of histories HH) if and only if all histories in the set which satisfy the first mode graph also satisfy the second mode graph (in which case we say that the second mode graph is a consequent of the first mode graph).

More precisely, $MG_1 \sim^{HH} MG_2$ iff $\forall H \in HH: H \text{ sat } MG_1 \Rightarrow H \text{ sat } MG_2$.

Mode Graph Isomorphism

Two mode graphs which have an equivalent structure (i.e., possess the same mathematical structure) will be said to be isomorphic.

Definition: Mode graph isomorphism

A mode graph MG_1 is said to be isomorphic to a mode graph MG_2 if there is a bijection $\alpha: M(MG_1) \rightarrow M(MG_2)$ such that the following conditions hold:

- i) (u, v) is in $A(MG_1)$ if and only if $(\alpha(u), \alpha(v))$ is in $A(MG_2)$, and*
- ii) w is an element of $S(MG_1)$ if and only if $\alpha(w)$ is an element of $S(MG_2)$ and x is an element of $E(MG_1)$ if and only if $\alpha(x)$ is an element of $E(MG_2)$.*

Such an α is called the isomorphism of MG_i onto MG_j . The isomorphism relation is denoted by $MG_i \cong(\alpha) MG_j$ (or simply $MG_1 \cong MG_2$).

The usual isomorphic relation for digraphs is defined in terms of a bijection only (i.e., condition i of mode graph isomorphism). However, for mode graph isomorphism an additional condition is imposed over the start and end modes. It should be noted that the satisfaction of the first condition does not imply the satisfaction of the second condition

this is illustrated by the examples in figure 4.4. We can construct a bijection from MG_1 to MG_2 that satisfies the condition i) of the mode graph isomorphism relation but no bijection can be constructed that satisfies conditions i) and ii). However, MG_1 and MG_3 are isomorphic mode graphs.

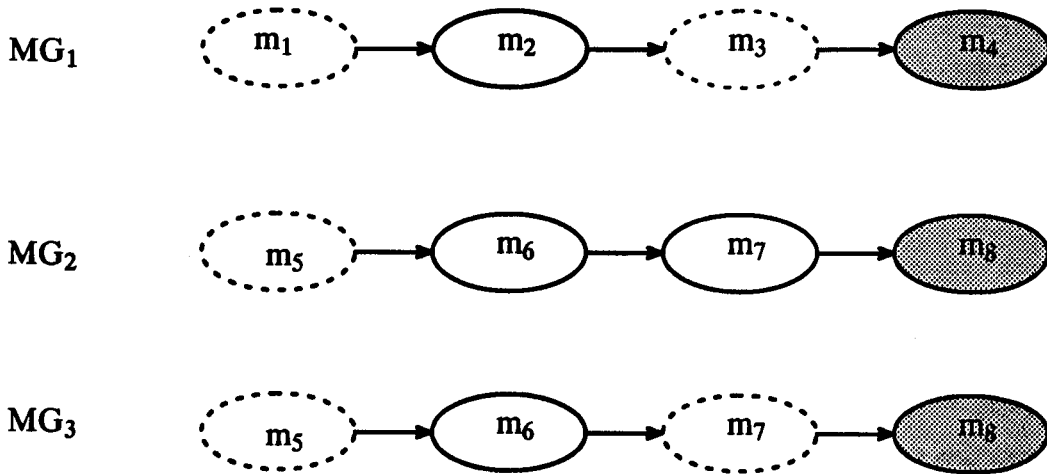


Figure 4.4. Mode Graph Isomorphism

Properties

The isomorphism relation is reflexive, symmetric and transitive. Thus the relation \simeq is an equivalence relation on a set of mode graphs. The equivalence classes are called isomorphism classes.

4.3.6. Mode Graph Categories

In this section I will discuss the constructs (which are based on mode graphs) that will be used to express some of the essential specifications produced during the analysis of a system. There are two variants of mode graphs a restricted class of mode graphs *single entry exit mode graphs*; and an informal version of a mode graph referred to as a *phase graph*.

For each construct I will describe the general characteristics of the specifications that can be expressed by it; and point out some interesting properties that can be derived from its structure. Detailed discussions of the role the constructs play during the analysis will be presented later. The roles of the constructs concerning the specifications produced during the real world analysis are discussed in chapter 5, and those produced during the controller

analysis in chapter 6. In these chapters the benefits gained from adopting the constructs will also be pointed out.

Single Entry Exit Mode Graphs

Single entry exit mode graphs (SEMG) are mode graphs for which there is a unique start mode and a unique end mode, and for which the start and end modes can occur only once in the mode sequence given by the graph. The class of behaviours that can be described by these graphs are those which have a unique start task and unique end task; and for which the start and end tasks are performed only once.

Definition: Single entry exit mode graph

A mode graph is a single entry exit mode graph if and only if the following conditions hold: i) there is only one mode in the start set; ii) there is only one mode in the end set; iii) the predecessor set of the start mode is empty; and iv) the successor set of the end mode is empty.

More precisely, a mode graph MG is a single entry exit mode graph iff:

- i) $|S(MG)| = 1$; ii) $|E(MG)| = 1$; iii) $\forall m \in S(MG): MG.pr(m) = \emptyset$
and iv) $\forall m \in E(MG): MG.sr(m) = \emptyset$.

To simplify analysis of single entry exit mode graphs the start and end modes of the graphs will be defined as modes (as opposed to sets); hence for an SEMG MG, $S(MG)$ and $E(MG)$ will return modes.

Lemma 4.10

From the conditions imposed on an SEMG we can infer the following graph-theoretic properties:

- i) *SEMGs are connected (from SEMG condition i and mode graph condition ii);*
- ii) *there is exactly one source node in an SEMG (from SEMG conditions i and iii and mode graph condition ii).*
- iii) *There is exactly one sink node in the graph (from SEMG conditions ii and iv and mode graph condition ii).*

Lemma 4.11

Any set of behaviours which can be represented by a mode graph can also be represented by an SEMG.

Proof

The lemma can be proven by the construction of an algorithm which modifies a mode graph (say, MG) to an equivalent SEMG. The algorithm is given below by the definition of the transformation Mtrans.

Algorithm 4.1

Function Mtrans(MG: ModeGraph): SEMGraph;

Var G: SEMGraph;

S, E: Modes;

1: S := $\langle \text{true}, \text{true}, \text{SStart}(S(\text{MG})) \rangle$;

2: E := $\langle \Omega, \text{true}, \Omega \rangle$;

3: M(G) := M(MG) $\cup \{S, E\}$;

4: A(G) := A(MG) $\cup \{(u, v) \in M(G) \times M(G) \mid (u = S \wedge v \in S(G)) \vee (u \in E(G) \wedge v = E)\}$;

5: S(G) := S;

6: E(G) := E;

7: Mtrans := G;

7: Stop.

Phase Graphs

A full formal analysis should not be conducted until preliminary work has identified the basic structure of the system and the variables of interest. This preliminary work follows the same pattern as formal analysis, but uses informal substitutes for modes and state variables. Thus instead of modes we work with *phases* which set out the same information, but in English (as opposed to mathematics). These phases are structured in phase graphs, which are a dual of mode graphs but with phases replacing modes. Hence, a phase graph can specify informally the same class of behaviours as a mode graph. More precisely, we define a phase graph as a four-tuple, $\text{PHG} = \langle \text{PH}, A, S, E \rangle$, where PH is a (finite) set of Phases, A is a set of phase pairs (i.e., $A \subseteq \text{PH} \times \text{PH}$), S and E are non-empty subsets of PH (i.e., $S, E \subseteq \text{PH}$, $S \neq \emptyset$ and $E \neq \emptyset$). The phase graph representation of the requirements (system concept) acts as a bridge between an informal requirements specification and a formal specification (expressed as a mode graph).

4.3.7. Predicate Mode Graph

In this section, I will introduce the concept, of a *predicate mode graph*, as a tool to simplify the verification of the real world specifications against the controller specifications. Predicate mode graphs are constructed for mode graphs that specify system behaviour at the controller level. Typically, a function is defined from the set of modes of a mode graph to a set of system predicates which are defined over the real world variables such that the system predicate given by applying the function to a mode is satisfied at the start of the mode.

Definition: Predicate mode graph

A predicate mode graph is represented as a pair $PG = \langle MG, PF \rangle$, where MG is an SEMG and PF is a function from the mode set of MG to a set of system predicates.

Predicate mode graphs will be used to represent a relationship between the satisfaction of the mode graph MG and the system conditions at the start point of the mode in MG . The relationship is captured by the definition of a complete predicate mode graph.

Definition: Complete predicate mode graph

A predicate mode graph $PG (= \langle MG, PF \rangle)$ is a complete predicate mode graph for a history description D and a system predicate SP if and only if:

i) for any pair (x, y) in $A(MG)$, any history H from $Set(D)$ and any interval Int , H satisfies $PF(x)$ at $s(Int)$ and x during Int and the start and invariant predicates of y at $e(Int)$ implies H satisfies $PF(y)$ at $e(Int)$; and

ii) for any history H and any time point t if H satisfies SP at t then H satisfies $PF(S(MG))$ at t .

The fact that a predicate mode graph PG is complete for a history description D and a system predicate SP will be denoted by $PG \text{ cmp } \langle D, SP \rangle$.

More precisely, $PG \text{ cmp } \langle D, SP \rangle$ iff

i) $\forall (x, y) \in A(MG): \forall H \in Set(D): \forall Int \in SI(T):$

$[H \text{ sat } PF(x)@s(Int) \wedge H \text{ sat } x@Int \wedge H \text{ sat } (Start(y) \wedge Inv(y)) @e(Int) \Rightarrow H \text{ sat } PF(y)@e(Int)]$

and ii) $\forall H \in Set(D): \forall t \in T: [H \text{ sat } SP@t \Rightarrow H \text{ sat } PF(S(MG))@t]$.

Complete Predicate Mode Graph Theorem

The complete predicate mode graph theorem (theorem 4.4) shows that if a predicate mode graph $\langle MG, PF \rangle$ is complete for a given history description D and system predicate SP , it follows that if a mode sequence ms of MG is satisfied by a history from $Set(D)$ for a sequence of time points $t_0, \dots, t_{|ms|}$ and SP satisfied at t_0 then precondition of mode $ms(i)$ is satisfied at the time point t_{i-1} , for $i = 0, \dots, |ms|$.

Theorem 4.4

If a predicate mode graph PG is complete for a history description D and a system predicate SP then for any history H from D and any ms of $Seq(MG(PG))$ if H satisfies ms for any sequence of time points $t_0, \dots, t_{|ms|}$ and SP satisfied at t_0 then $PF(ms(i))$ is satisfied at the time point t_i , for $i = 0, \dots, |ms|$.

$PG \text{ cmp } \langle D, SP \rangle \rightarrow$

$\forall H \in Set(D): \forall ms \in Seq(MG): \forall t_0, \dots, t_{|ms|} \in T:$

$[H \text{ sat } ms@ \langle t_0, \dots, t_{|ms|} \rangle \wedge H \text{ sat } SP@t_0 \rightarrow \forall i \in \{1, \dots, |ms|\}: H \text{ sat } PF(ms(i))@t_{i-1}].$

Proof. (By contradiction).

Assume:

$PG \text{ cmp } \langle D, SP \rangle \wedge$

$\exists H \in Set(D): \exists ms \in Seq(MG): \exists t_0, \dots, t_{|ms|} \in T:$

$[H \text{ sat } ms@ \langle t_0, \dots, t_{|ms|} \rangle \wedge H \text{ sat } SP@t_0 \wedge \exists i \in \{1, \dots, |ms|\}: H \text{ sat } \neg PF(ms(i))@t_{i-1}].$

Firstly, we make the observation that since $ms(1) = S(MG)$ from clause *ii* of $PG \text{ cmp } \langle D, SP \rangle$, it follows that $H \text{ sat } PF(ms(1))@t_0$.

If $i = 1$, that is, $H \text{ sat } \neg PF(ms(1))@t_0$ we have a contradiction. If $i > 1$, since $H \text{ sat } ms(i-1)@ [t_{i-2}, t_{i-1}]$, from $H \text{ sat } \neg PF(ms(i))@t_{i-1}$ and clause i of $PG \text{ cmp } \langle D, SP \rangle$, it follows that $H \text{ sat } \neg PF(ms(i-1))@t_{i-2}$. Hence if $i > 1$, it follows that $H \text{ sat } \neg PF(ms(1))@t_0$, which leads to a contradiction.

4.4. Summary

This chapter, introduced the concept of a *mode* as a construct which can be used to specify the behaviour that must be exhibited by the system during an interval for it to perform the specified task. The formal semantics of a mode were specified in terms of a

history and the satisfaction conditions (discussed in chapter three). Several examples were used to illustrate how modes can be used to specify tasks. The notion of a consistent mode was introduced and the conditions to determine the consistency were presented. *Mode sequences* were introduced as a means to compose modes; and the notion of mode consistency was defined.

A graphical notation called a *mode graph* was used to represent the transition between modes. It was shown how a mode graph defines a set of mode sequences; then the semantics of the mode graph were defined in terms of the set. The notion of a complete mode graph was introduced, as a mode graph for which at the end point of the satisfaction of a mode a successor of that mode must be start satisfied. Consistent mode graphs were defined as those mode graphs for which the all the modes in its sequence set are consistent.

To simplify the comparison of mode graphs the notion of *predicate mode graphs* was introduced. Predicate mode graphs are used to augment mode graphs, that specify behaviour at the controller level, with the behaviour that is exhibited at the real-world level at the start of the modes – for a history that satisfies the mode graph. Predicate mode graphs are used in verification strategies discussed in chapter 8.

This chapter (and chapter three) has set up the apparatus (i.e., the formal model) that will be used during the requirements analysis of a system. In the remainder of this thesis a role for the formal model and system development model (chapter two) during the requirements analysis will be explained in detail.

Chapter 5 - Real World Specifications

In this chapter the essential specifications that are produced during the real world analysis are discussed; for each specification its role in the description of system behaviour is discussed. These roles are exemplified by presenting the essential specifications of the reaction vessel.

5.1. Disaster Set

A disaster of a system is formulated as a Boolean state variable. The disaster, represented by the state variable, is said to occur in a system history if and only if there is a time point during the system lifetime at which the system predicate (given by the state variable) is satisfied, for that history. The disaster set of a system is the set of all of the disasters (i.e. state variables that represent disasters) of the system. The disaster set of a system SY, is denoted by $\text{Dis}(\text{SY})$. The disaster predicate of a system is the system predicate given by the disjunction of the disasters of the system. The disaster predicate of a system SY, is denoted by $\text{Dip}(\text{SY})$. The role of the disaster set of a system is to represent all potential disasters of that system as a set of system predicates, to facilitate a systematic analysis of the hazards of the system. The role of the disaster predicate of a system is to represent all system conditions in which a disaster has occurred.

Definition: Disaster-free histories

The set of disaster free histories of a system SY (denoted by $\text{DFH}(\text{SY})$) is the set of all histories that satisfy the negation of the disaster predicate of the system.

More precisely, $\text{DFH}(\text{SY}) = \text{def} \{H \in \Gamma H \mid H \text{ sat } \neg \text{Dip}(\text{SY})\}$.

Example Disaster Set

Let us suppose that a disaster analysis of the reaction vessel shows that there is a risk of an explosion. In the formal framework there must therefore be a system predicate representing this disaster (i.e., the explosion). The simplest approach is to introduce a new state variable to directly label this. For the reaction vessel we have p_{20} as the elementary predicate representing an explosion. Since the disaster analysis indicates that there are no

other potential disasters we have $\text{Dis}(\text{Rv}) = \{p_{20}\}$ and $\text{Dip}(\text{Rv}) = p_{20}$.

The formalization of the disasters of a system provides formal markers which label the potential disasters of the system, on their own the disaster sets are of limited use in the safety analysis, we are more concerned with the system conditions (hazards) that can lead to the disasters. However, the main reason for introducing the disaster set is that, the knowledge of the disasters allows us to focus the hazard analysis on the potential disasters of the system.

Catastrophe Class

For a state variable that represents a disaster, we wish to assert the property that once that start variable is satisfied for a particular history H at a time point t it will be satisfied for all time points after t for H . To formally capture this property we introduce the notion of a catastrophe class.

Definition: Catastrophe variable

A catastrophe variable is Boolean variable which is true at the start point of the system lifetime, right continuous and can never be false once it becomes true.

More precisely, if p_i is a catastrophe variable then:

$$p_i(s(T)) = \text{true} \wedge \forall t_0 \in T: \lim_{t \rightarrow t_0^+} p_i(t) = p_i(t_0) \wedge \forall t \in T: [p_i(t) \Rightarrow \forall t' > t: p_i(t')]$$

5.2. Safety Real World Description

The safety real world description of a system is formulated as a history description over the safety real world variables; for a system SY it is denoted by $\text{SRD}(\text{SY})$. We define the set of safety real world description histories ($\text{SRDH}(\text{SY})$) as: $\text{SRDH}(\text{SY}) = \text{Set}(\text{SRD}(\text{SY}))$. The role of the safety real world description is to specify the behaviour of the (real world) environment that impinges on safety-critical behaviour of the system, by using the description relations.

Example Safety Real World Description

The class relations of the safety real world description of the reaction vessel are specified by table 5.1. The tables are obtained by a careful systematic analysis of the

environment of the system at the real world level which focuses on the safety-critical behaviour of the system (the analysis process is detailed in chapter 7).

The safety real world description is defined as: $\text{SRD}(\text{Rv}) = \langle \text{T}, \text{Sv}, \text{VP}, \text{CP}, \text{IR}, \text{HR} \rangle$, where

$$\text{Sv} = \langle p_1, \dots, p_{19} \rangle;$$

$$\text{VP} = \langle \text{Vp}_1, \dots, \text{Vp}_{19} \rangle;$$

$$\text{CP} = \langle \text{Cp}_1, \dots, \text{Cp}_{19} \rangle;$$

$$\text{IR} = \langle \text{Ir}_1, \text{Ir}_2, \text{Ir}_3 \rangle \text{ and } \text{HR} = \langle \text{Hr}_1 \rangle.$$

The safety real world description history set is defined as: $\text{SRDH}(\text{Rv}) = \text{Set}(\text{SRD}(\text{Rv}))$.

Table 5.1: Relations of Safety Real World Description

No.	Related variables	Relationship	Comments
Ir ₁	p ₇ , p ₈ , p ₉	$p_9 = p_7 + p_8$	The flow rate into the vessel is the sum of FlowA and FlowB.
Ir ₂	p ₁₄ , p ₁₅ , p ₁₆	$p_{16} = p_{14} + p_{15}$	The flow rate out of the vessel is the sum of OutflowC and OutflowD.
Ir ₃	p ₁ , p ₁₃ , p ₁₉	$p_1 = s(\text{T}) \Rightarrow p_{13} = 0$	At the start of the system lifetime the vessel is empty.
Hr ₁	p ₆	$p_{6,1} \leq p_{6,0} + \Delta \text{Tm} \times \text{dur}$	ΔTm is the maximum rise in temperature per second.

5.3. Hazard Specification

The hazards of a system are normally also characterized by predicates [Gors86, Leve87]. With this approach a hazard is present during a history of the system if the predicate that characterizes the hazard is satisfied at some time point during the history. Such hazards can be captured in the specification model by system predicates. The role of the hazard specification of a system is to represent all system conditions (i.e., hazards) that can lead to a disaster, as a system predicate over the real world variables

For example, consider the following hazard in system EX: “An explosion may occur if the concentration of X in a mixing vessel (denoted by p_x) is greater than the concentration of Y (denoted by p_y)”. We can easily express this with the predicate: $p_x > p_y$. The hazard which is characterized by $p_x > p_y$ is said to be present in a history H iff $\exists t \in \text{T}: \text{H sat } p_x > p_y @ t$.

The identification of hazards will normally be performed by considering the set of all possible disasters. We will define the hazards of a disaster as a system predicate which specifies all the (identified) conditions over a system history that can lead to the disaster; for a disaster x we call this predicate the hazard predicate of x and denote it by $HZ(x)$. Roughly speaking, such a system predicate $HZ(x)$ is complete for the disaster x , if for any possible history H the disaster x is satisfied at a time point t only if $HZ(x)$ is satisfied at some time point prior to t . That is a hazard is a prerequisite of a disaster. Of course, for a particular system there is no way of being certain that all the hazards of a disaster have been identified. The statement that a hazard predicate of a given disaster is complete, is in essence an assumption that all the hazards of that disaster have been captured by the hazard predicate. The notion of the hazard of a disaster can be extended to cover all disasters and this leads to the definition of a hazard specification.

Definition: Hazard specification

The hazard specification of a system is a system predicate which specifies all the (identified) conditions over a system history which can lead to any disaster. For a system SY this will be denoted by $HS(SY)$.

Definition: Complete hazard specification assumption

For any possible history and any time point t , if the disaster predicate is satisfied at t then H must satisfy the hazard specification at some time point prior to t .

More precisely, for any possible history H ,

$$\forall t \in T: [H \text{ sat Dip}(SY)@t \Rightarrow \exists t' \in [s(T), t): H \text{ sat } HS(SY)@t']$$

Hence, we define the set of histories for which the above assumption holds as follows:

$$HA(SY) =$$

$$\{H \in \Gamma H(SY) \mid \forall t \in T: [H \text{ sat Dip}(SY)@t \Rightarrow \exists t' \in [s(T), t): H \text{ sat } HS(SY)@t']\}.$$

We will refer to the set of histories $HA(SY)$ as the hazard analysed histories of the system SY . The complete hazard assumption is a formal statement of the assumption that the hazard analysis has identified all possible hazards, for the disaster set of the system.

Lemma 5.1

For any history H from $HA(SY)$ if H satisfies the negation of the hazard specification of SY then

H does not satisfy the disaster predicate of SY.

More precisely, $\forall H \in HA(SY): H \text{ sat } \neg HS(SY) \rightarrow H \text{ sat } \neg Dip(SY)$.

Proof. By contradiction.

Assume: $\exists H \in HA(SY): H \text{ sat } \neg HS(SY) \wedge \exists t \in T: H \text{ sat } Dip(SY)@t$.

From the definition of $HA(SY)$ we have:

$\exists t \in T: H \text{ sat } Dip(SY)@t \rightarrow \exists t' \in [s(T), t): H \text{ sat } HS(SY)@t'$.

This contradicts the antecedent $H \text{ sat } \neg HS(SY)$.

The consequence of lemma 5.1, is that if the absence of hazardous states can be ensured during a hazard analysed history, then a disaster cannot occur during that history.

Lemma 5.2

For any history H from HA(SY) if for all time points t H satisfies the negation of the disaster predicate during $[s(T), t]$ implies H satisfies negation of the hazard specification during $[s(T), t]$ then H satisfies the negation of the hazard specification.

More precisely,

$\forall H \in HA: [\forall t \in T: H \text{ sat } \neg Dip(SY)@[s(T), t] \rightarrow H \text{ sat } \neg HS(SY)@[s(T), t] \rightarrow H \text{ sat } \neg HS(SY)]$.

Proof.

Firstly, we make the observation that this lemma holds if the following condition holds.

$\forall H \in HA: [\forall t \in T: H \text{ sat } \neg Dip(SY)@[s(T), t] \rightarrow H \text{ sat } \neg HS(SY)@[s(T), t] \rightarrow H \text{ sat } \neg Dip(SY)]$.

Next, we prove the above by contradiction.

Assume: $\exists H \in HA: [\forall t \in T: H \text{ sat } \neg Dip(SY)@[s(T), t] \rightarrow H \text{ sat } \neg HS(SY)@[s(T), t] \wedge \exists t \in T: H \text{ sat } Dip(SY)@t]$.

If $Dip(SY)$ is satisfied at a time point t , there must be a time up to (and including) t at which it is satisfied for the first time.

$\exists t \in T: H \text{ sat } Dip(SY)@t \rightarrow \exists t' \in [s(T), t]: H \text{ sat } Dip(SY)@t'$.

From the definition of event satisfaction we have:

$H \text{ sat } Dip(SY)@t' \rightarrow H \text{ sat } \neg Dip(SY)@[s(T), t']$.

From $\forall t \in T: H \text{ sat } \neg Dip(SY)@[s(T), t] \rightarrow H \text{ sat } \neg HS(SY)@[s(T), t]$ we have:

$H \text{ sat } \neg \text{Dip}(SY)@[s(T), t'] \rightarrow H \text{ sat } \neg HS(SY)@[s(T), t].$

Since H is from the set HA we have:

$H \text{ sat } \text{Dip}(SY)@[s(T), t'] \rightarrow \exists t'' \in [s(T), t']: H \text{ sat } HS(SY)@t''.$

But this leads to a contradiction.

The consequence of lemma 5.2, is that for any history from HA , if assuming the absence of a disaster up to any time point implies the absence of a hazardous state up to that time point, then a hazardous state cannot be present for that history.

Once the complete hazard assumption has been confirmed for a system. We adopt the convention that the complete hazard assumption holds for the history sets of the history descriptions of the system. This convention avoids the necessity to explicitly state the complete hazard assumption for each history set.

Hazard Specification Example

Let us suppose that the hazard analysis for the disaster of the reaction vessel identifies the following (simple) hazard. An explosion may occur if the temperature of the contents of the vessel rises above a specified value ($Eact \text{ } ^\circ K$) during the reaction. This hazard can be specified as: $p_6 \geq Eact \wedge p_{13} \neq 0$.

5.4. Safety Real World Specification

The role of the safety real world specification is to specify the conditions that must be maintained during a history for that history to be free of any of the (identified) disasters. The safety real world specification for a system SY is denoted as $SRS(SY)$ and is formulated as a system predicate imposed over the safety real world variables of the system.

Definition: Safety real world histories

The set of safety real world histories of a system SY is the subset of safety real world description histories of a system that satisfy the safety real world description. This set will be denoted by $SRH(SY)$.

More precisely, $SRH(SY) = \{H \in SRDH(SY) \mid H \text{ sat } SRS(SY)\}.$

For a system SY the safety real world specification $SRS(SY)$ is defined as the negation of the hazard specification $HS(SY)$ (i.e., $SRS(SY) = \neg HS(SY)$). For any system SY clearly any system history H, under the hazard specification assumption, that satisfies the safety real world specification (as defined above) is a disaster free history.

Example Safety Real World Specifications

In the section below, we present the safety real world specifications of the example hazards from section 5.3.

For the system EX we have, $HS(EX) = p_x > p_y$, thus $SRS(EX) = p_x \leq p_y$.

For the system Rv we have, $HS(Rv) = p_6 \geq Eact \wedge p_{13} \neq 0$, thus

$SRS(Rv) = p_6 < Eact \vee p_{13} = 0$.

5.5. Mission Real World Description

The mission real world description of a system is formulated as a history description over the mission real world variables; for a system SY it is denoted by $MRD(SY)$. We define the mission real world description histories ($MRDH(SY)$) as: $MRDH(SY) = Set(MRD(SY))$. The role of the mission real world description is to specify behaviour of the (real world) environment that impinges on the mission-oriented behaviour of the system, by using the description relations.

Example Mission Real World Description

The relations of the mission real world description are specified by table 5.2. Table 5.2 is obtained by a careful systematic analysis of the environment of the system at the real world level which focuses on the mission-oriented behaviour of the system (the analysis process is detailed in chapter 7).

The mission real world description is defined as: $MRD(Rv) = \langle T, Sv, VP, CP, IR, HR \rangle$, where

$Sv = \langle p_1, \dots, p_{18} \rangle$;

$VP = \langle Vp_1, \dots, Vp_{18} \rangle$;

$CP = \langle Cp_1, \dots, Cp_{18} \rangle$;

$IR = \langle Ir_1, \dots, Ir_5 \rangle$; and $HR = \langle Hr_1, \dots, Hr_5 \rangle$.

The mission real world description set is defined by MRD: $MRDH(R_v) = \text{Set}(MRD(R_v))$.

Table 5.2: Relationships of Mission Real World Description

No.	Related variables	Relationship	Comments
Ir ₁	p ₇ , p ₈ , p ₉	$p_9 = p_7 + p_8$	The flow rate into the vessel is the sum of FlowA and FlowB.
Ir ₂	p ₁₄ , p ₁₅ , p ₁₆	$p_{16} = p_{14} + p_{15}$	The flow rate out of the vessel is the sum of OutflowC and OutflowD.
Ir ₃	p ₁ , p ₁₃ , p ₁₉	$p_1 = s(T) \Rightarrow p_{13} = 0$	At the start of the system lifetime the vessel is empty.
Ir ₄	p ₁₀ , p ₁₁ , p ₁₂ , p ₁₃	$p_{13} = p_{10} + p_{11} + p_{12}$	The volume of liquid in the vessel is the sum of the volumes of A, B and C.
Ir ₅	p ₁ , p ₁₃ , p ₁₈	$p_1 = S(T) \Rightarrow p_{18} = g$	At the start of the system lifetime the indicator is at green.
Hr ₁	p ₆	$p_{6,1} \leq p_{6,0} + \Delta T_m \times \text{dur}$	ΔT_m is the maximum rise in temperature per second.
Hr ₂	p ₂ , p ₃ , p ₄ , p ₅	$\forall t: p_5(t) = \text{start} \vee p_5(t) = \text{collect} \Rightarrow$ $\forall t: p_2(t) = p_{2,0} \wedge p_3(t) = p_{3,0} \wedge$ $p_4(t) = p_{4,0}$	The set point selectors must remain constant while the Plant select is at start or collect.
Hr ₃	p ₆ , p ₇ , p ₁₀ , p ₁₂ , p ₁₆	$\forall t: p_6(t) < M_{act} \wedge p_{16}(t) = 0 \wedge p_{12,0} = 0$ $\Rightarrow p_{10,1} = \text{Min}(\int p_7 dt + p_{10,0}, V_{max})$	Provided the temperature remains below M_{act} , the outflow rate is zero during an interval, and at the start of the interval there is no C in the vessel, then VolA is equal to the smaller of the sum of the volume the integral of the flow rate over the interval and the maximum level.
Hr ₄	p ₆ , p ₈ , p ₁₁ , p ₁₂ , p ₁₆	$\forall t: p_6(t) < M_{act} \wedge p_1(t) = 0 \wedge p_{12,0} = 0$ \Rightarrow $p_{11,1} = \text{Min}(\int p_8(t) dt + p_{11,0}, V_{max})$	This relation is the VolB equivalent of the above relation.
Hr ₅	p ₉ , p ₁₃ , p ₁₆	$\forall t: p_9(t) = 0 \Rightarrow$ $p_{13,1} = p_{13,0} - \int p_{16}(t) dt$	Provided the flow rate is zero during an interval the volume of liquid in the vessel at the end of the interval is the volume at the start of the interval minus the integral of outflow over the interval.

5.6. Mission Real World Specification

The mission real world specification of a system SY is formulated as an SEMG; and for a system SY it is denoted by MRS(SY). The system predicates used to construct the mode specification of a mission real world specification are imposed over the mission real world variables. We define the *mission real world histories* of a system SY (denoted by MRH(SY))

as the set of mission real world description histories that satisfy the mission real world specification (i.e., $MRH(SY) = \{H \in MRDH(SY) \mid H \text{ sat } MRS(SY)\}$). The role of the mission real world specification of a system SY is to specify the behaviour that must be exhibited by the (mission) real world variables for the system to fulfil its mission, as an SEMG.

5.6.1. Mission Phase Specification

The role of the mission phase specification is to structure the system concept, before a detailed formal analysis is performed. The mission phase specification of a system SY is expressed by a phase graph, and denoted by $MPS(SY)$. The formal analysis is performed over the mission phase specification by studying the behaviour of each of the phases; the behaviour expressed by the informal specification of the phases is then formally expressed by an SEMG. As a result of the formal analysis a formal description for each phase of the phase graph is defined. An approach which supports the construction of a mission real world specification from a mission phase specification is presented in chapter 7.

Example Mission Real World Specification

In this section I will show how the mission of the reaction vessel can be formally expressed as an SEMG. Recall that the mission real world variables for the reaction vessel are: $\langle p_1, \dots, p_{16}, p_{18} \rangle$. The structure of the mission real world specification and the mode specifications are produced by a systematic analysis of the mission phase specification and the system concept (the general process is discussed in chapter 7).

Mission Real World Specification Structure

The structure of the mission requirements specification of the reaction vessel is:

$M = \{\text{Closed, Set points, Set up, Amber, Activate, Production, Collect, Operator, End}\};$

$A = \{(\text{Closed, Set points}), (\text{Set points, Set up}), (\text{Set up, Amber}), (\text{Amber, Activate}), (\text{Activate, Production}), (\text{Production, Collect}), (\text{Collect, Operator}), (\text{Operator, Set points}), (\text{Operator, End})\};$

$S = \text{Closed};$ and $E = \text{End}$. The picture is given in figure 5.1.

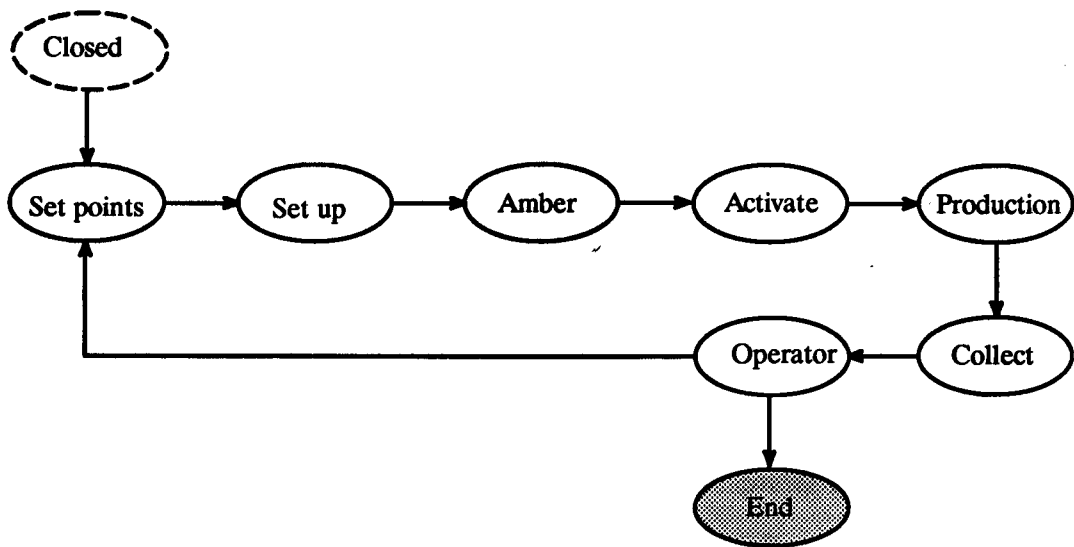


Figure 5.1. Reaction Vessel Mission Real World Specification Structure

Mode Specifications

The specifications of the modes of the reaction vessel (mission specification) are given in the following paragraphs.

Closed Mode

The system enters the closed mode at the start of the system lifetime. During this mode, the vessel must be empty and the Indicator must be green. The system must leave this mode as soon as the Plant select is turned to on.

$\text{Closed} = \langle \text{true}, p_{13} = 0 \wedge p_{18} = g, p_5 = \text{on} \rangle.$

Set points Mode

The system is in this mode while the operator selects the required set points. At the start of the mode the plant select is at on. During the mode, Plant select must be at on or start, the vessel must be empty and the Indicator must be green. The system must leave this mode as soon as Plant select is turned to start.

$\text{Set points} = \langle p_5 = \text{on}, p_5 \in \{\text{on}, \text{start}\} \wedge p_{13} = 0 \wedge p_{18} = g, p_5 = \text{start} \rangle.$

Set up Mode

The system is in this mode while the required volumes of A and B are being loaded into the vessel. During this mode, Plant select must be at start, Temperature must be less than the minimum activation temperature or the vessel must be empty and the Indicator green. The

system must leave this mode as soon as the volumes of A and B are both approximately equal to their set point values. The system must spend at most $f_{ST}(p_2, p_3)$ seconds in the mode, where f_{ST} is a function which defines an upper bound on the time taken to fill the vessel with the required volumes.

Set up = $\langle \text{true}, p_5 = \text{start} \wedge (p_6 < \text{Mact} \vee p_{13} = 0) \wedge p_{18} = g, \text{Dvol}, 0, f_{ST}(p_2, p_3) \rangle$,
where $\text{Dvol} = (|p_{10} - p_2| \leq \Delta v_A \wedge |p_{11} - p_3| \leq \Delta v_B)$.

Amber Mode

The system is in this mode while the Indicator is being set to amber. During this mode, Plant select must be at start, the Temperature must be less than the minimum activation temperature, the volumes of A and B must be approximately at the set point values and the Indicator must be at green or amber. The system must leave this mode as soon as the Indicator is at amber. The system must spend at most ΔA seconds in the mode.

Amber = $\langle \text{true}, p_5 = \text{start} \wedge p_6 < \text{Mact} \wedge \text{Dvol} \wedge p_{18} \in \{g, a\}, p_{18} = a, 0, \Delta A \rangle$.

Activate Mode

The system is in this mode when the temperature of the vessel is being raised to the activation temperature. During this mode, Plant select must be at start, the Temperature of the vessel must be less than (or equal to) the set point value plus a small tolerance (ΔT_v), no liquid must enter or leave the vessel and the Indicator must be at amber or red. The system must leave this mode as soon as the Temperature of the vessel is within the activation temperature range (i.e. the temperature is greater than or equal to the set point value, but less than the set point value plus ΔT_v) and the Indicator is at red. The system must spend at most $f_{RT}(p_4)$ seconds in this mode, where $f_{RT}(p_4)$ is a function that defines an upper bound on the time taken to raise the temperature to its set point value.

Activate = $\langle \text{true}, \text{Inv}, \text{ST} \wedge p_{18} = r, 0, f_{RT}(p_4) \rangle$,

where $\text{Inv} = (p_5 = \text{start} \wedge p_6 \leq p_4 + \Delta T_v \wedge p_9 = 0 \wedge p_{16} = 0 \wedge p_{18} \in \{a, r\})$;

and $\text{ST} = (0 \leq p_6 - p_4 \leq \Delta T_v)$.

Production Mode

The system is in this mode while the reaction is taking place. During this mode, Plant select must be at start or collect, the Temperature of the vessel is within the activation range, no liquid

must leave or enter the vessel and the Indicator must be at red. The system must leave this mode as soon as the Plant select is turned to collect.

$\text{Production} = \langle \text{true}, p_5 \in \{\text{start}, \text{collect}\} \wedge \text{ST} \wedge p_9 = 0 \wedge p_{16} = 0 \wedge p_{18} = r, p_5 = \text{collect} \rangle.$

Collect Mode

The system is in this mode while the product is being collected. During this mode Plant select must at collect, the Temperature of the vessel must be within the activation temperature range and the Indicator must be at red, or the vessel must be empty, and the Indicator must be at red or green. The system must leave this mode as soon as the vessel is empty and the Indicator is at green. The system must spend at most ΔC seconds in this mode.

$\text{Collect} = \langle \text{true}, \text{Inv}, p_{13} = 0 \wedge p_{18} = g, 0, \Delta C \rangle,$

where $\text{Inv} = (p_5 = \text{collect} \wedge ((\text{ST} \wedge p_{18} = r) \vee p_{13} = 0) \wedge p_{18} \in \{g, r\})$.

Operator Mode

The system is in this mode while the operator decides whether another batch of C is to be produced or operation is to be ceased. During this mode Plant select must be at collect, off or on, the vessel must be empty and the Indicator must be at green. The system must leave this mode as soon as the Plant select is at off or on.

$\text{Collect} = \langle \text{true}, p_5 \in \{\text{collect}, \text{off}, \text{on}\} \wedge p_{13} = 0 \wedge p_{18} = g, p_5 \in \{\text{off}, \text{on}\} \rangle.$

End Mode

The system is in this mode when no more C will be produced. At the start of the mode the Plant select is at off. During this mode, the vessel must be empty and the indicator must be at green.

The system remains in this mode for the remainder of the system lifetime.

$\text{End} = \langle p_5 = \text{off}, p_{13} = 0 \wedge p_{18} = g, \Omega \rangle.$

5.7. Summary

The chapter discussed the specification model in the context of the specifications produced during the real world analysis. The formal representations of the specifications produced during the analysis, were presented. The concepts introduced during the chapter were illustrated by several examples – the real world specifications of the reaction vessel (introduced in chapter two) were given in full.

It was shown how the *disasters* of a system can be represented as a set of system predicates and the *safety real world description* encoded as a history description. The *hazard specification* and *safety real world specification* were also expressed as system predicates. The safety-critical behaviour of a system SY at the real world level is expressed as a pair $\langle \text{SRD}(\text{SY}), \text{SRS}(\text{SY}) \rangle$.

It was shown how the *mission real world description* is encoded as a history description. The formal representation of the *mission real world specification*, as an SEMG was discussed. The concept of a *mission phase specification* was introduced as informal (but structured) representation of the system concept, that is expressed as a phase graph. The mission-oriented behaviour of a system SY at the real world level is expressed as a pair $\langle \text{MRD}(\text{SY}), \text{MRS}(\text{SY}) \rangle$.

The clarification of the role of the essential specifications produced during the real world analysis; and the identification of the relationships between the constructs of the formal model and these essential specifications provides a useful support to the real world analysis. However, for the analysts to improve the confidence of the analysts that the essential specifications produced during the real world analysis fulfil their roles, a related systematic methodology is required for the productions of these specifications. A suitable methodology should exploit the general structure of the constructs which will be used to express the real world specifications.

Chapter 6 - Controller Specifications

In this chapter the essential specifications that are produced during the controller analysis are discussed; for each specification its role in the description of system behaviour is given. These roles are exemplified by presenting the essential specifications of the reaction vessel.

6.1. Safety Environment Description

The role of the safety environment description of a system to represent the relationship between the behaviour of the sensors and actuators of the safety controller of a system and the real world variables, in addition to any relations defined by the safety real world description of that system, by using the description relations. The safety environment description is defined, in terms of a history description; for a system SY this is denoted by SED(SY). The safety environment description is produced as an extension to the safety real world description. To the the sequence of safety real world variables we add the sequence of safety controller variables. The ranges and classes of the safety controller variables are added to the range sequence and class sequence of the safety real world description. Any invariant (resp. history) relations involving the safety controller variables are added to the invariant (resp. history) relation sequence of the safety real world description. We define the safety environment description histories of a system SY (denoted by SEDH(SY)) as: $SEDH = \text{Set}(SED(SY))$.

Example Safety Environment Description

The safety environment description of the reaction vessel is an extension of the safety real world description of the reaction vessel (see section 5.2). The relations involving the safety controller variables and real world variables are defined by table 6.1. The table is obtained by a systematic analysis of the properties of the sensors and actuators of the safety controller (the general analysis process is detailed in chapter 8).

The safety environment description is defined as: $SED(Rv) = \langle T, Sv, VP, CP, IR, HR \rangle$, where

$$Sv = \langle p_1, \dots, p_{24} \rangle;$$

$VP = \langle Vp_1, \dots, Vp_{24} \rangle$; $CP = \langle Cp_1, \dots, Cp_{24} \rangle$;

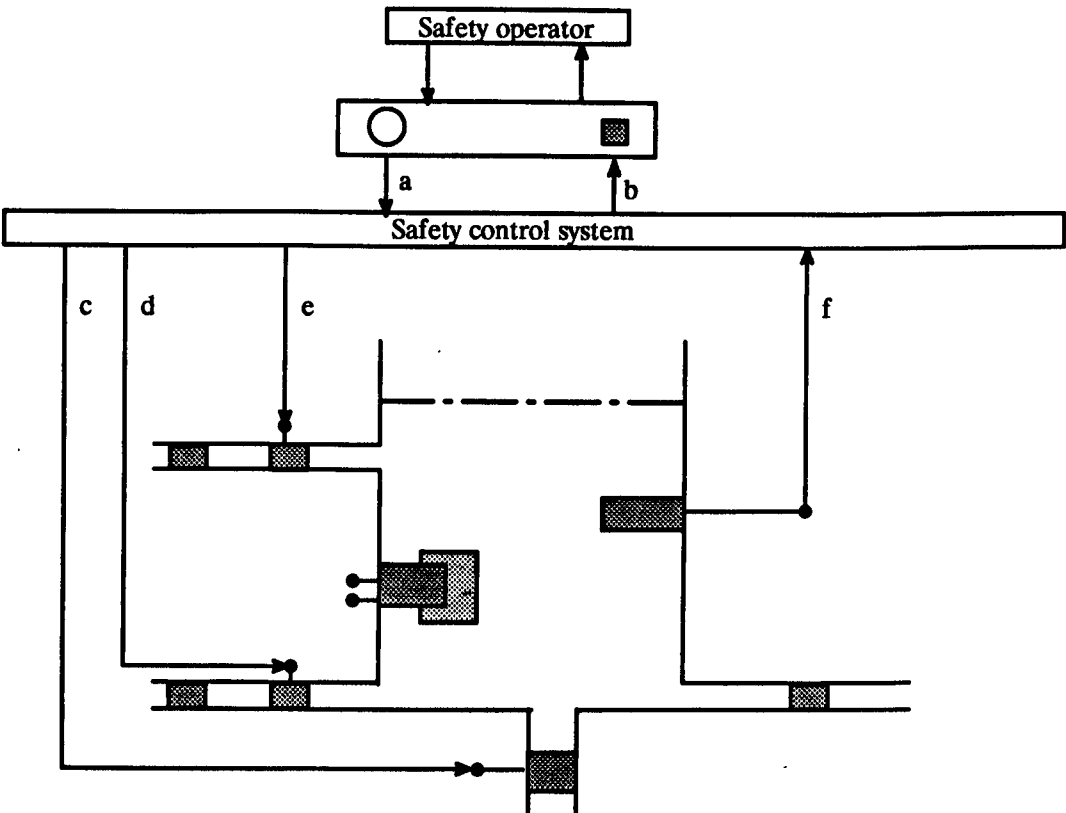
$IR = \langle Ir_1, Ir_2, Ir_3, Ir_4, Ir_6, Ir_7, Ir_8 \rangle$ and $HR = \langle Hr_1, Hr_6 \rangle$.

The safety environment description history set is defined as: $SRDH(Rv) = Set(SRD(Rv))$.

Table 6.1: Controller Relationships of Safety Environment Description

No.	Related variables	Relationship	Comments
Ir ₆	p ₇ , p ₂₁	$p_{21} = \text{on} \Rightarrow p_7 = 0$.	If LockA is on, FlowA is zero
Ir ₇	p ₈ , p ₂₂	$p_{22} = \text{on} \Rightarrow p_8 = 0$	If LockB is on, FlowB is zero.
Ir ₈	p ₆ , p ₂₄	$ p_{24} - p_6 \leq \Delta Tp$	The inaccuracy in the Thermometer is bounded by ΔTp .
Hr ₆	p ₉ , p ₁₃ , p ₂₃	$\text{dur} \geq ET \wedge$ $\forall t: p_9(t) = 0 \wedge p_{23}(t) = D_{\text{max}}$ \Rightarrow $p_{13,1} = 0$.	If no liquid is flowing into the vessel and ValveD is open (to its full width) for an interval of duration greater than ET then the vessel will be empty at the end of the interval.

The safety controller is illustrated in figure 6.1.



Key

a: Safety select b: Safelight c: ValveD d: LockB e: LockA f: Thermometer

Figure 6.1. Reaction Vessel Safety Controller

6.2. Safety Controller Specification

The role of the safety controller specification is to specify a behaviour over the safety controller variables that will ensure (under the safety controller description) that the conditions expressed by the safety real world specification are maintained. The safety controller of a system SY is formally expressed as an SEMG, and denoted by $SCS(SY)$. The system predicates used to construct the mode specifications of a safety controller specification are imposed over the safety controller variables. These mode specifications express the behaviour that the control system (i.e., the computing system) must ensure is exhibited by the safety controller. The invariants of these mode specifications must be imposed only over those safety controller variables over which the control system or safety operator has direct control, since the control system or safety operator must be able to ensure that these invariants are satisfied during the modes. In particular, an invariant must not be imposed over the safety controller variables that represent sensors, since the control system has no direct control over the value of a sensor (the value of a sensor is governed by the value of the real world variable being sensed). We define the *safety controller histories* of a system SY (denoted by $SCH(SY)$) as the set of safety environment description histories that satisfy the safety controller specification (i.e., $SCH(SY) = \{H \in SEDH(SY) \mid H \text{ sat } SCS(SY)\}$).

6.2.1. Safety Controller Behaviour Structure

To guide the analysis and verification of the safety controller specification, a general structure is proposed for the behaviour of any safety controller. The structure is represented by a phase graph (see figure 6.2).

The phase graph can be characterized by the following general phase sequence:

$\langle \text{start up, monitor, } \langle \text{recovery, reset, monitor} \rangle^n, \text{recovery}^m, \text{shut down, end} \rangle$,

where $n > 0$, $m \in \{0, 1\}$.

In the following sections I will give a description of the role of each phase in the behaviour of any safety controller, and discuss the formal constructs that will be used to specify the behaviour of the safety controller during these phases.

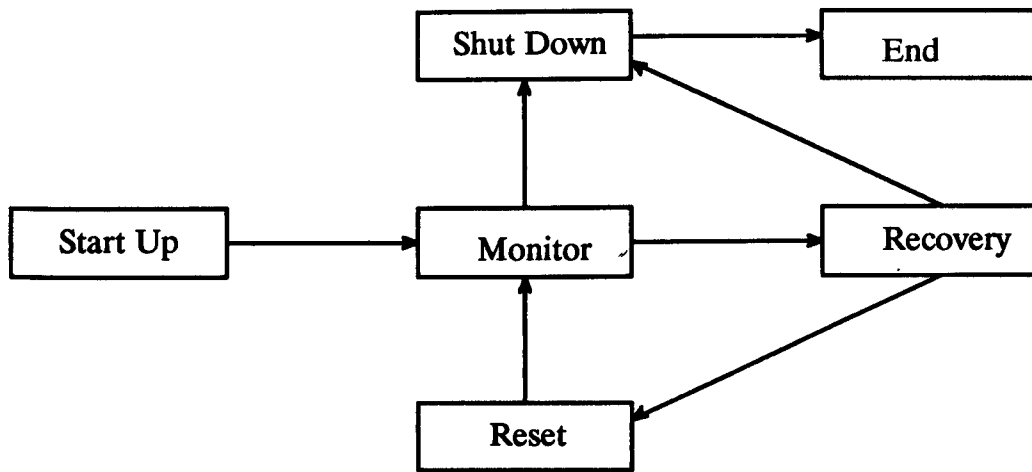


Figure 6.2. General Safety Controller Specification Structure

Example Safety Controller Specification

To illustrate how the general structure can be used to specify the behaviour of a safety controller, the specifications of the reaction vessel will be given for each phase. Recall that, in addition to the behaviour specified by the safety real world specification (see chapter 5), the behaviour of the safety controller should comply with the (informal) requirements on the Safelight.

Safelight Requirements

The informal requirements over the Safelight can be restated in terms of the general phases of the safety controller.

- a. The Safelight should be *green* when the safety controller resides in the end phase;
- b. The Safelight should be *green or amber* when the safety controller resides in the start up or reset phases;
- c. The Safelight should be *amber* when the safety controller resides in the monitor phase; and
- d. the Safelight should be *red* soon after the safety controller enters the recovery phase or shut down phase; and remain at *red* until the safety controller has completed the recovery or shut down actions at which point the Safelight should be green.

Specification Condition

In the safety environment description of the reaction vessel, constants were used in the specification of description relations. For example, in the history relation Hr_6 :

$$[\text{dur} \geq ET \wedge \forall t: p_9(t) = 0 \wedge p_{23}(t) = D_{\max} \wedge \text{dur} \geq ET] \Rightarrow p_{13,1} = 0,$$

the constant ET is used to define an upper bound on the time taken to empty the vessel when the flow rate is zero and ValveD is open. Such constants will be referred to as description constants.

The formulation of several mode specifications is dependent on the value of the description constants. For such mode specifications, constants and functions are used in the formulation of the system predicates and time bounds. These constants and functions are related to the description constants by specification conditions. For example, the mode specification of the empty mode (see later) is defined using the constants L_1 and U_3 :

$$\text{Empty} = \langle p_{24} < \text{temp}_5, p_{19} \in \{g, r\} \wedge \forall \text{set}, p_{19} = g, L_1, U_3 \rangle,$$

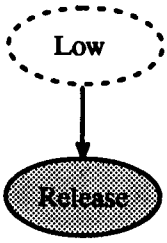
The specification condition for L_1 ensures that the safety controller stays in the mode long enough to empty the vessel so we must have: $L_1 > ET$.

6.2.2. Start Up Phase

The start up phase of a system lifetime is the interval during which the safety controller is set up. Typically, during the start up phase the safety controller will set up the initial conditions of the physical process and set the safety controller variables to their initial values. Roughly speaking, the initial conditions of the physical process are chosen to ensure that the system cannot enter into a hazardous state for an interval after the start up phase. The system enters the start up phase at the start of its lifetime and leaves when the start event occurs. The required behaviour of the safety controller during the start up phase should be specified by an SEMG, referred to as the start up graph (denoted by $SU(SY)$).

Example Start Up Graph

The start up graph, of the reaction vessel, is a simple SEMG of two connected modes. The structure is drawn below and the modes are specified in the following paragraphs.



$$SU(Rv) = \langle \{Low, Release\}, (Low, Release), Low, Release \rangle.$$

Low Mode

The safety controller starts in this mode. During this mode, Safelight is green, LockA and LockB must be on, and ValveD must be closed. The safety controller leaves this mode as soon as the Safety select is turned on; and the Thermometer is less than (or equal to) $temp_1$.

$$Low = \langle true, p_{19} = g \wedge p_{21} = on \wedge p_{22} = on \wedge p_{23} = 0, p_{17} = on \wedge p_{24} \leq temp_1 \rangle.$$

Release Mode

The safety controller is in this mode while it is setting up the actuators for the monitor phase. At the start of the mode the Thermometer must be less than (or equal to) $temp_1$. During this mode Safelight is green or amber and ValveD must be closed. The safety controller must turn Safelight to amber, Release LockA and LockB; and then leave this mode. The safety controller must not spend at most U_1 seconds in the mode.

$$Release = \langle p_{24} \leq temp_1, p_{19} \in \{g, a\} \wedge p_{23} = 0, p_{19} = a \wedge p_{21} = off \wedge p_{22} = off, 0, U_1 \rangle.$$

Specification condition

The temperature during the release mode must be below the minimum activation temperature.

So $temp_1$ and U_1 must be chosen in accordance with the inequality:

$$temp_1 + \Delta T_m \cdot U_1 + \Delta T_e < Mact - \Delta T_p.$$

6.2.3. Monitor Phase

A monitor phase is an interval of the system lifetime during which the safety controller monitors the behaviour of the physical process. Typically, during a monitor phase the safety controller will monitor the behaviour of the physical process to detect any recovery events. Roughly speaking a recovery event is an event after which the safety controller must enter

the recovery phase. The system enters a monitor phase after the start up phase or after the reset phase (see later); it leaves when a recovery or shut down event occurs. During the monitor phase the behaviour of the actuators of the safety controller are specified by a system predicate – the monitor invariant of the controller – denoted by $\text{MonInv}(\text{SY})$. The behaviour of the safety controller during a monitor phase is specified by an SEMG referred to as the monitor graph (denoted by $\text{MN}(\text{SY})$).

Example Monitor Graph

The required behaviour of the safety controller of the reaction vessel is specified by a trivial graph: $\text{MN}(\text{Rv}) = \langle \{\text{Monitor}\}, \emptyset, \text{Monitor}, \text{Monitor} \rangle$.

Monitor Mode

The safety controller is in this mode during the monitor phase. At the start of the mode the Thermometer reading must be less than temp_2 . During this mode Safelight must be at amber, LockA and LockB must be released; and ValveD must be closed. The safety controller must leave this mode as soon as the Safety select is at off or the Thermometer reading is greater than (or equal to) temp_3 .

$\text{Monitor} = \langle p_{24} < \text{temp}_2, \text{MInv}, p_{17} = \text{off} \vee p_{24} \geq \text{temp}_3 \rangle$,

where $\text{MInv} = (p_{19} = a \wedge p_{21} = \text{off} \wedge p_{22} = \text{off} \wedge p_{23} = 0)$.

Specification condition

The thermometer reading at the end of the release mode must comply with the start predicate of the monitor mode, the start predicate of the monitor mode must imply the negation of the end predicate of the monitor mode; and the temperature during the monitor mode must be below the explosion temperature.

So temp_2 and temp_3 must be chosen in accordance to the inequality:

$$(\text{temp}_1 + \Delta T_m \cdot U_1 + \Delta T_e < \text{temp}_2) \wedge (\text{temp}_2 \leq \text{temp}_3) \wedge (\text{temp}_3 + \Delta T_e < E_{\text{act}} - \Delta T_p).$$

6.2.4. Recovery Phase

A recovery phase of a system lifetime is an interval during which the safety controller takes over the main control of the system in order to prevent the system from entering into a hazardous state. Typically, during a recovery phase the safety controller will manipulate

its actuators to avoid a hazard. A system history enters a recovery phase from a monitor phase when the safety controller detects a recovery event and leaves when a shutdown or reset event occurs. A recovery event can occur in any mode of the monitor graph; and the recovery events of different modes can initiate different recovery actions. Hence, a safety controller could have a recovery graph for each mode of the monitor graph. The behaviour of the safety subsystem during a recovery phase is specified by a set of SEMGs, referred to as recovery graphs. The recovery graphs are denoted by the function $REC(SY)$, where $REC(m)$ gives the recovery graph of mode m . The function $REC(SY)$ will be used in the definition of a function which connects the recovery graphs to the safety controller specification (see chapter 8).

Example Recovery Graph

The safety controller enters the recovery phase when there is risk of the temperature becoming too high. Let us suppose that both the collection vessel for OutletC and the collection vessel for OutletD maintain the temperature of their contents below $Eact$. (In the analysis of a real system the previous assumptions would have to be added to the safety real world description by introducing variables for the temperature and volume of liquid in the collection vessels.) Hence, provided the vessel can be emptied before the temperature reaches $Eact$ the hazard will be avoided.

The recovery function for the reaction vessel is defined as: $REC(Rv)(Monitor) = \langle \{Detect, Empty, Cooling\}, \{(Detect, Empty), (Empty, Cooling)\}, Detect, Cooling \rangle$.



Detect Mode

The safety controller enters this mode after the recovery condition of the safety controller has been detected, and remains in it until the actuators are set up for a recovery. At the start of this mode the Thermometer reading is less than $temp_4$. The safety controller must turn Safelight to red, lock LockA and LockB, open ValveD; and then leave this mode. The safety controller must spend at most U_2 seconds in the mode.

Controller Specifications

Detect = $\langle p_{24} < \text{temp}_4, \text{true}, p_{19} = r \wedge p_{21} = \text{on} \wedge p_{22} = \text{on} \wedge p_{23} = D_{\text{max}}, 0, U_2 \rangle$.

Specification condition

The thermometer reading at the end of the monitor mode must imply the start predicate of the detect mode; and the temperature during the detect mode must be below the explosion temperature.

So temp_4 and U_2 must be chosen in accordance to the inequality:

$$(\text{temp}_3 + \Delta T_e < \text{temp}_4) \wedge (\text{temp}_4 + \Delta T_m.U_2 + \Delta T_e < E_{\text{act}} - \Delta T_p).$$

Empty Mode

The safety controller is in this mode while the vessel is being emptied, via OutletD. At the start of the mode the Thermometer reading is less than temp_5 . During this mode, Safelight must be at green or red, LockA and LockB must be locked, and ValveD must be open. The safety controller must turn the Safelight to green, and then leave this mode. The safety controller must spend between L_1 and U_3 seconds

$$\text{Empty} = \langle p_{24} < \text{temp}_5, p_{19} \in \{g, r\} \wedge V_{\text{set}}, p_{19} = g, L_1, U_3 \rangle,$$

$$\text{where } V_{\text{set}} = (p_{21} = \text{on} \wedge p_{22} = \text{on} \wedge p_{23} = D_{\text{max}}).$$

Specification conditions

i) *The thermometer reading at the end of the detect mode must imply the start predicate of the empty mode; and the temperature during the empty mode must be below the explosion temperature. So temp_5 and U_3 must be chosen in accordance to the inequality:*

$$(\text{temp}_4 + \Delta T_m.U_2 + \Delta T_e < \text{temp}_5) \wedge (\text{temp}_5 + \Delta T_m.U_3 + \Delta T_e < E_{\text{act}} - \Delta T_p).$$

ii) *The safety controller must stay in the mode long enough to ensure that the mode is empty.*

So L_1 must be chosen in accordance to the inequality: $L_1 > ET$.

Cooling Mode

The safety controller is in this mode while the vessel is cooling down after it has been emptied. During this mode, the Safelight is green, LockA and LockB are locked and ValveD is open. The safety controller must leave this mode as soon as the Safety select is at off or the Thermometer reading is less than temp_6 and the Safety select is at reset.

$$\text{Cooling} = \langle \text{true}, p_{19} = g \wedge V_{\text{set}}, p_{17} = \text{off} \vee (p_{24} < \text{temp}_6 \wedge p_{17} = \text{reset}) \rangle.$$

6.2.5. Reset Phase

A reset phase is an interval of the system lifetime which occurs after a recovery phase after which the safety controller is allowed to return to the monitor phase. That is, during the reset phase the state of the real world changes to a state from which the safety controller can enter the monitor phase, with the assurance that a recovery condition will not be satisfied at the instant the safety controller enters the monitor phase. The reset phase is introduced to improve the availability of the system rather than system safety. The safety controller enters a reset phase when a reset event occurs and leaves when a monitor event occurs. Typically, during a reset phase the safety controller will reset the actuators to allow the mission controller to proceed with the mission, by returning to the monitor phase.

The safety controller may also perform safety checks before entering into the monitor phase if any of the checks are not passed the controller will remain in the reset phase. A safety controller may enter the reset phase from the end mode of any recovery graph. The behaviour of the safety subsystem during the reset phase is specified by a set of SEMGs, referred to as reset graphs. The reset graphs are denoted by the function $RG(SY)$, where $RG(SY)(m)$ gives the reset graph of mode m . The function $RG(SY)$ is used in the definition of an algorithm which connects the recovery graphs to the safety controller specification (see chapter 8).

Example Reset Graph

The reset function is defined over the cooling mode as follows:

$$RG(Rv)(cooling) = \langle \{Reset\}, \emptyset, Reset, Reset \rangle.$$

Reset Mode

The safety controller is in this mode while valves are being reset. At the start of the mode Safety select is at reset and the Thermometer is less than $temp_6$. During this mode, Safelight must be at green or amber. The safety controller must turn Safelight to green, release LockA and LockB close ValveD, and then leave this mode. The safety controller must spend at most U_4 seconds in this mode.

Reset =

$\langle p_{17} = \text{reset} \wedge p_{24} < \text{temp}_6, p_{19} \in \{g, a\}, p_{19} = g \wedge p_{21} = \text{off} \wedge p_{22} = \text{off} \wedge p_{23} = 0, 0, U_4 \rangle$.

Specification Condition

The thermometer reading at the end of the reset mode must comply with the start predicate of the monitor mode. So temp_6 and U_4 be chosen in accordance to the inequality:

$\text{temp}_6 + \Delta T_m \cdot U_4 < \text{temp}_2$.

6.2.6. Shut Down Phase

A shut down phase of a system is an interval of the system lifetime after which the safety controller can no longer enter the monitor phase. Typically, at the end of the shut down phase, the state of the real world is such that provided the value of the safety controllers actuators are unchanged the system cannot enter into a hazardous state. The safety controller enters the shut down phase when a shut down event occurs and leaves when an end event occurs. A safety controller may enter the shutdown phase from the end mode of any recovery graph or from a monitor mode. The behaviour of the safety subsystem during the shutdown phase is specified by SEMGs, referred to as *shut down graphs*. The shut down graphs are denoted by the function Sh , where $Sh(m)$ gives the shut down graph of mode m . The function Sh is used in the definition of an algorithm which connects the shutdown graphs to the safety controller specification (see chapter 8).

Example Shut Down Graphs

The shutdown graph of the cooling mode is the empty graph, since no tasks need be performed after the recovery to allow the controller to be shut down (i.e., the vessel is empty and ValveA and ValveB are locked). The shut down graph of the monitor mode is simply the recovery graph.

6.2.7. End Phase

The end phase of a system is the interval of the system lifetime after a shutdown phase up to the end of the systems lifetime. The safety controller enters the end phase after an end event occurs and leaves when the termination predicate is satisfied. The behaviour of

the safety controller during the phase is specified by an unbounded mode, referred to as a end controller mode (denoted by EC(SY)).

Example End Controller Mode

End control Mode

The safety controller is in this mode during the end phase of the safety controller. At the start of the mode the Safety select is at reset and the vessel is empty. During this mode, Safelight must be green, LockA and LockB must be locked and ValveD must be open. The safety controller leaves this mode as soon as the termination predicate holds.

End control = $\langle p_{17} = \text{reset}, p_{19} = g \wedge V_{\text{set}}, \Omega \rangle$.

Safety Controller Specification Example

The formal structure of the safety controller specification of the reaction vessel is:

$M = \{\text{Low, Release, Monitor, Detect, Empty, Cooling, Reset, End Control}\};$

$A = \{(\text{Low, Release}) (\text{Release, Monitor}), (\text{Monitor, Detect}), (\text{Detect, Empty}), (\text{Empty, Cooling}), (\text{Cooling, Reset}), (\text{Cooling, End Control})\};$

$S = \text{Low};$ and $E = \text{End Control}.$

A diagram of the safety controller specification structure is given in figure 6.3.

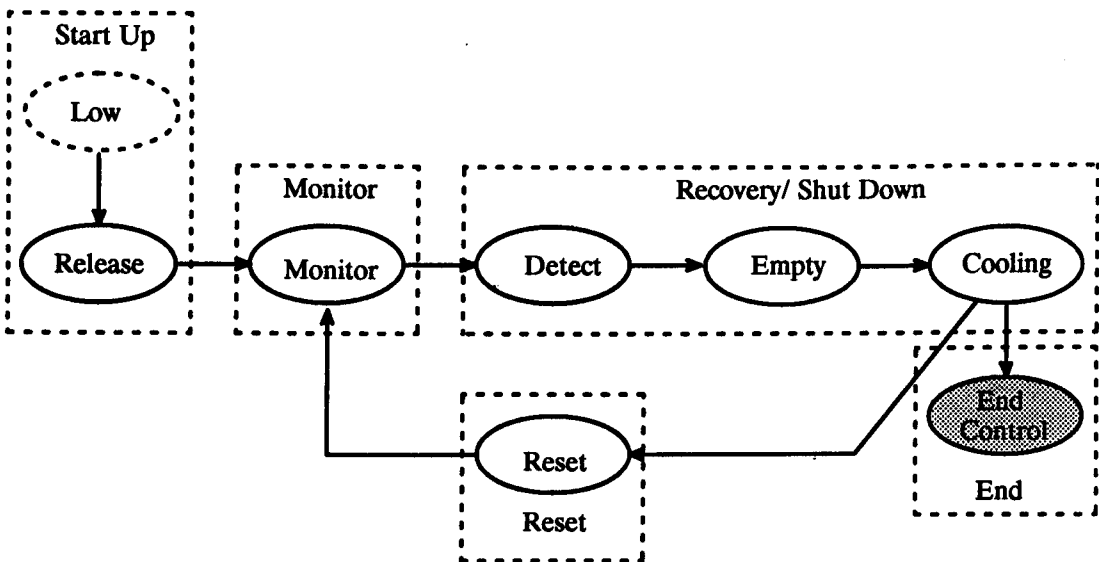


Figure 6.3. Reaction Vessel Safety Controller Specification Structure

6.3. Mission Environment Description

The role of the mission environment description of a system is to represent the relationship between the behaviour of the sensors and actuators of the safety controller of a system and the real world variables, in addition to any relations defined by the mission real world description of that system, by using the description relations. The mission environment description is defined, in terms of a history description; for a system SY this is denoted by $MED(SY)$. The mission environment description is produced as an extension to the mission real world description. To the sequence of mission real world variables we add the sequence of mission controller variables and some safety controller variables (roughly speaking these are the safety controller variables which have an influence on the effect of the mission controller on the real world, referred to as the external controller variables). The ranges and classes of the mission controller variables and external controller variables are added to the range sequence and class sequence of the mission real world description. Any invariant (resp. history) relations involving the mission controller variables, external controller variables and real world variables are added to the invariant (resp. history) relation sequence of the mission real world description. We define the mission environment description histories of a system SY (denoted by $MEDH(SY)$) as: $MEDH(SY) = \text{Set}(MED(SY))$.

6.3.1. Relation Classes

One of the main reasons for partitioning the controller (into safety and mission controllers) was to minimise any relationships between the mission-oriented and safety-critical functions of the controller. For the systems considered, relationships involving the mission controller and safety controller variables will be severely restricted – by defining two sorts of relations: mission and dependent.

Mission Relations

A mission relation is a description relation that is specified over the real world variables and the mission controller variables only. Clearly, a mission relation cannot describe any relationship between the safety and mission controller variables.

Dependent Relations

For some systems it will not be possible to express all relevant relationships as mission relations. In particular, a relationship between the behaviour of mission controller variables and real world variables may be dependent on the value of some safety controller variables. A class of relations which can capture this property is referred to as dependent relations, these are defined next.

Definition: Dependent relation

A dependent relation is an invariant relation formulated using two system predicates SP and MP, as $SP \Rightarrow MP$. SP is defined over the safety controller variables and MP is defined over the mission variables. The value of the predicate SP must be under the control of the safety control system, and the predicate MP describes a relationship between a mission controller variable and the real world which cannot be controlled by the mission control system.

As an example of a dependent relation consider the following relation of the reaction vessel: if LockA is released FlowA is given by a function over ValveA, this is captured by the following dependent relation $p_{21} = \text{off} \Rightarrow p_7 = f_{IA}(p_{25})$.

Example Mission Environment Description

The mission environment description of the reaction vessel is given as an extension of the mission real world description (see section 5.5). The relations involving the mission controller variables are defined by table 6.2. This table is obtained by a systematic analysis of the properties of the mission controllers actuators and sensors (the general analysis process is detailed in chapter 8).

The mission environment description is defined as: $MED(Rv) = \langle T, Sv, VP, CP, IR, HR \rangle$, where

$Sv = \langle p_1, \dots, p_{29} \rangle$; $VP = \langle Vp_1, \dots, Vp_{29} \rangle$; $CP = \langle Cp_1, \dots, Cp_{29} \rangle$;

$IR = \langle Ir_1, \dots, Ir_{13} \rangle$ and $HR = \langle Hr_1, \dots, Hr_5, Hr_7, Hr_8, Hr_9 \rangle$.

The mission environment description history set is defined as: $MEDH(Rv) = \text{Set}(MED(Rv))$.

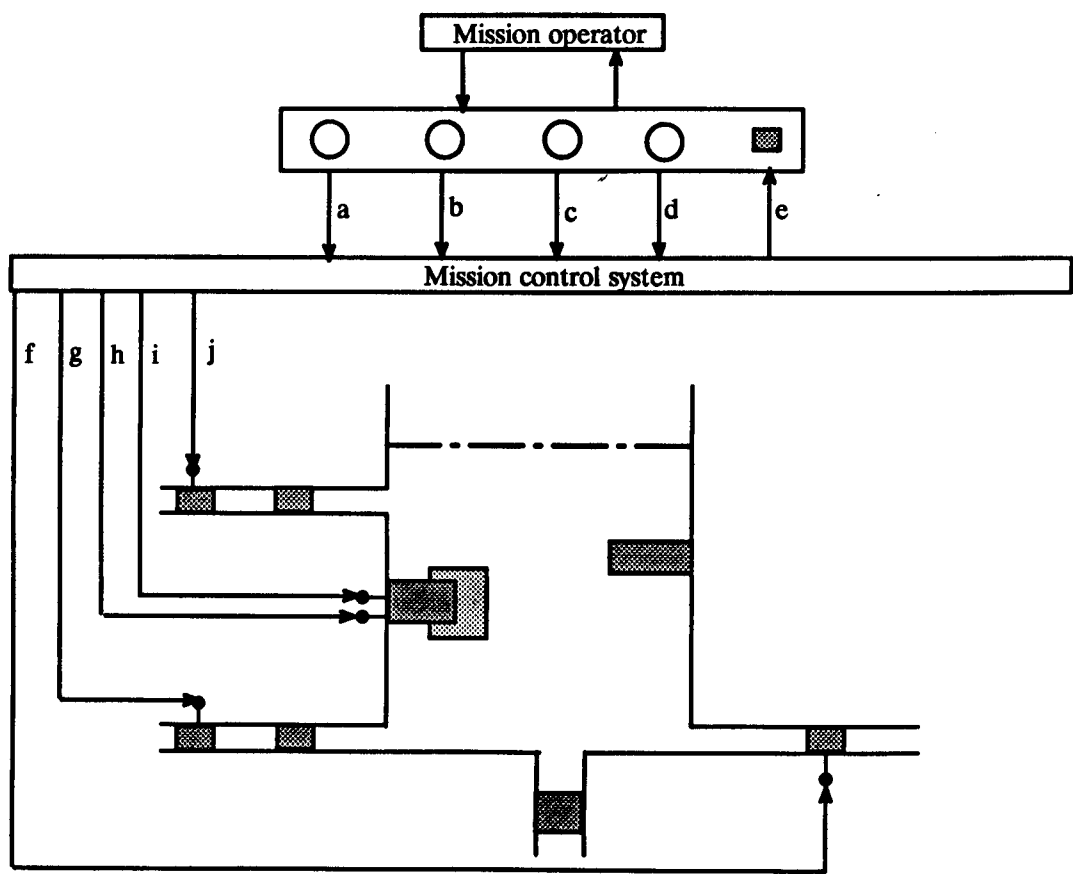
Table 6.2: Controller Relationships of Mission Environment Description

No.	Related variables	Relationship	Comments
Ir9	p7, p21, p25	$p_{21} = \text{off} \Rightarrow p_7 = f_{1A}(p_{25})$	If LockA is released FlowA is given by a function over ValveA.
Ir10	p8, p22, p26	$p_{22} = \text{off} \Rightarrow p_8 = f_{1B}(p_{26})$	If LockB is released FlowB is given by a function over ValveB.
Ir11	p13, p15, p27	$p_{14} = f_{OC}(p_{13}, p_{27})$	OutflowC is given by a function over Volume and ValveC.
Ir12	p6, p13, p28	$p_{13} = 0 \wedge p_{28} = \text{off} \Rightarrow p_6 < \text{Mact}$	If the vessel is empty and the regulator of then the temperature is less than the minimum activation value.
Ir13	p18, p25, p26, p27, p28	$p_1 = s(T) \Rightarrow$ $p_{18} = g \wedge p_{25} = 0 \wedge p_{26} = 0 \wedge$ $p_{27} = 0 \wedge p_{28,0} = \text{off}.$	At the start of the system lifetime the indicator is at green ValveA, ValveB and ValveC are closed; and the regulator is off.
Hr7	p6, p28, p29	$\forall t: (p_{28} = \text{on} \wedge p_{29} = p_{29,0})$ $\wedge \text{dur} > \text{Rt}(p_{29})$ $\Rightarrow \forall t::(\Delta \text{Rt}(p_{29}), 0): p_{29} - p_6 \leq \Delta \text{Reg})$	If the regulator is on for an interval during which regulator set is constant and the duration of the interval is greater than the value given by Rt(p29) then after the first Rt(p29) seconds of the interval the temperature is approximately the set value.
Hr8	p6, p28	$p_{6,0} \leq \text{Mact} \wedge \forall t: p_{28}(t) = \text{off} \Rightarrow$ $\forall t: p_6(t) \leq \text{Mact}$	If the temperature is below Mact at the start of the interval and the regulator is off during the interval then the temperature is below Mact during the interval.
Hr9	p13, p25, p26, p27	$\forall t: [p_{25}(t) = 0 \wedge p_{26}(t) = 0 \wedge$ $p_{27}(t) = \text{Cmax}] \wedge \text{dur} \geq \text{ETC} \Rightarrow$ $p_{13,1} = 0$	If the inlet valves are closed and ValveC is open for an interval of duration at least ETC then the vessel will be empty at the end of the interval.

The mission controller is illustrated in figure 6.4.

6.3.2. Monitor Relations

For safety-critical systems the customer is concerned with the satisfaction of the mission requirements only when the safety controller does not override the mission controller. The safety controller does not override the mission controller when it is in the monitor phase. In the approach presented here, the mission controller need only satisfy the mission while the safety controller is in the monitor phase. Hence, any relations which hold during the monitor phase can be included in the mission environment description.



Key

a: VolA select b: VolB select c: Temp select d: Plant select e: Indicator
f: ValveC g: ValveB h: Regswitch i: Regtemp j: ValveA.

Figure 6.4. Reaction Vessel Mission Controller

Example Monitor Relations

For the reaction vessel we can add the invariant relation which follows from the monitor invariant of SCS(Rv) to MED(Rv). That is we add the invariant relation Ir₁₄: $p_{21} = \text{off} \wedge p_{22} = \text{off} \wedge p_{23} = 0$. The main consequence of adding this invariant relation is that for the set of histories described by the modified MED(Rv), the mission controller has full control over the flow rates into and out of the vessel.

6.4. Mission Controller Specification

The role of the mission controller specification of a system SY is to specify the behaviour that must be exhibited by the mission controller that will ensure that the mission

real world specification of the system will be satisfied. The mission controller specification of a system SY is expressed formally as an SEMG, and for a system SY it is denoted by $MCS(SY)$. The system predicates used to construct the mode specifications of a mission controller specification are imposed over the mission controller variables. These mode specifications express the behaviour that the control system and the mission operator must ensure is exhibited. The invariants of these mode specifications must be imposed only over those mission controller variables over which the control system or operator has direct control. As for the safety controller specification an invariant must not be imposed over the sensor variables. We define the *mission controller histories* of a system SY (denoted by $MCH(SY)$) as the set of mission environment histories that satisfy the mission controller specification (i.e., $MCH(SY) = \{H \in MEDH(SY) \mid H \text{ sat } MCS(SY)\}$).

6.4.1. Mission Controller Behaviour Structure

The strategy for the construction of the mission controller specification is based on a systematic analysis of the modes of the mission real world specification. For each mode of the mission real world specification, an SEMG is constructed that specifies the behaviour that must be exhibited by the mission controller to perform the task specified by the mode of the mission real world specification; such an SEMG is called the *controller graph* of the mode. As a result of this analysis a function is developed from the mode set of the mission real world specification to a set of SEMGs – the *controller function*. The mission controller specification would then be constructed by combining the graphs obtained from the controller function. A technique for the analysis of the mission controller based on the strategy outlined above is presented in chapter 8.

Mission Controller Specification Example

The mission controller specification of the reaction vessel is presented next. A relationship between the mission controller specification is also presented by defining the controller function of the reaction vessel. In this section, the mission controller specification is simply presented. A methodology to construct and check this specification is detailed in chapter 8.

Mission Controller Specification Structure

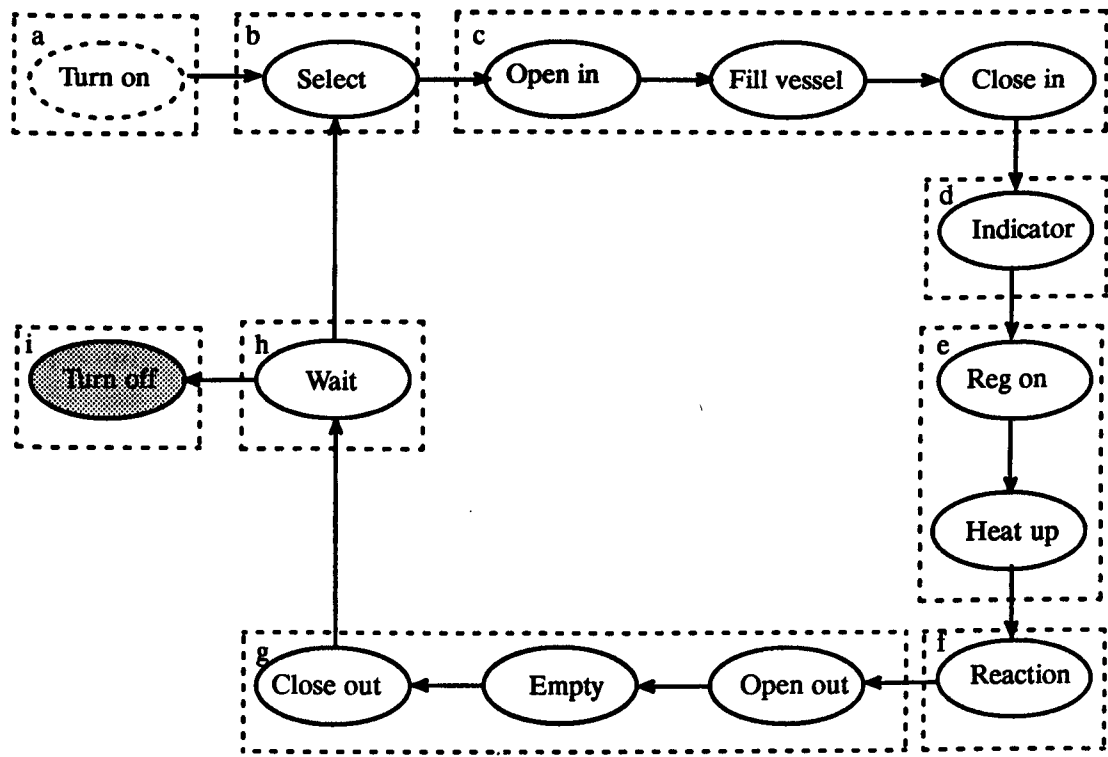
The structure of the mission controller specification is: $MCS(R_v) = \langle M, A, S, E \rangle$.

$M = \{ \text{Turn on, Select, Open in, Fill vessel Close in, Indicator, Reg on, Heat up, Reaction, Open out, Empty, Close out, Wait, Turn off} \};$

$A = \{ (\text{Turn on, Select}), (\text{Select, Open in}), (\text{Open in, Fill vessel}), (\text{Fill vessel, Close in}), (\text{Close in, Indicator}), (\text{Indicator, Reg on}), (\text{Reg on, Heat up}), (\text{Heat up, Reaction}), (\text{Open out, Empty}), (\text{Empty, Close out}), (\text{Close out, Wait}), (\text{Wait, Select}), (\text{Wait, Turn off}) \};$

$S = \text{Turn on}$ and $E = \text{Turn off}$.

The diagram of the mission controller specification is given in figure 6.5. The boxes in the diagram illustrate the relationship between the modes of the mission real world specification and the mission controller specification (i.e., the controller function).



Key

a: Closed	b: Set points	c: Set up	d: Amber	e: Activate
f: Production	g: Collect	h: Operator	i: End.	

Figure 6.5. Reaction Vessel Mission Controller Specification Structure

Mode Specifications

The modes used in the mission controller specification of the reaction vessel are presented in the following paragraphs. The specification conditions play the same role as they did for the safety controller specification

Closed Mode

$$CF(\text{Closed}) = \text{Turn on}$$

Turn on Mode

This is the start mode for the mission controller specification. During this mode, the Indicator must be at green; ValveA, ValveB and ValveC must be closed and the regulator must be off. The mission controller must leave this mode as soon as Plant select is at on.

$$\text{Turn on} = \langle \text{true}, p_{18} = g \wedge V_c \wedge p_{28} = \text{off}, p_5 = \text{on} \rangle,$$

$$\text{where } V_c = (p_{25} = 0 \wedge p_{26} = 0 \wedge p_{27} = 0).$$

Set points Mode

$$CF(\text{Set points}) = \text{Select}$$

Select Mode

The mission controller is in this mode while the operator selects the set points. At the start of the mode, Plant select is at on. During this mode, Plant select must be at on or start, the Indicator must be at green; ValveA, ValveB and ValveC must be closed and the regulator must be off. The mission controller must leave this mode as soon as Plant select is at start.

$$\text{Select} = \langle p_5 = \text{on}, p_5 \in \{\text{on}, \text{start}\} \wedge p_{18} = g \wedge V_c \wedge p_{28} = \text{off}, p_5 = \text{start} \rangle.$$

Set up Mode

$$CF(\text{Set up}) = \text{Open in} \rightarrow \text{Fill vessel} \rightarrow \text{Close in}$$

Open in Mode

Controller Specifications

This is the mode in which the valves are opened to fill the vessel with liquids A and B. During this mode, Plant select must be at start, the Indicator must be at green, ValveC must be closed and the regulator must be off. The mission controller must open ValveA to the extent required by the function f_A and ValveB to the extent required by the function f_B and then leave this mode. The mission controller must spend at most OT units of time in the mode.

Open in = $\langle \text{true}, F_v, O_v, 0, OT \rangle$,

where $F_v = (p_5 = \text{start} \wedge p_{18} = g \wedge p_{27} = 0 \wedge p_{28} = \text{off})$;

and $O_v = (p_{25} = f_A(p_2, p_3) \wedge p_{26} = f_B(p_2, p_3))$.

Specification Condition

The increase in the volume of A (resp. B) during the open in mode must be bounded by half of the volume A (resp. volume B) tolerance. So we must choose OT to satisfy the inequality: $f_{IA}(A_{\max}).OT < \Delta V_A/2 \wedge f_{IB}(B_{\max}).OT < \Delta V_B/2$.

Fill vessel Mode

The mission controller is in this mode while liquid A and liquid B are being loaded into the vessel. At the start of this mode, ValveA and ValveB must be open to the extent required by the functions f_A and f_B . During this mode, Plant select must be at start, the Indicator must be at green, ValveC must be closed, and the regulator must be off. The mission controller must start to close the inlet valves; and then leave this mode. The mission controller must spend between $f_T(p_2, p_3)$ and $f_T(p_2, p_3) + OT$ seconds in this mode.

Fill vessel = $\langle O_v, F_v, \neg O_v, f_T(p_2, p_3), f_T(p_2, p_3) + OT \rangle$.

Specification Condition

The volume of A (resp. volume of B) in the vessel at the end of the fill vessel mode must be at least the set point value minus the volume A (resp. volume of B) tolerance.

So we must define the functions f_A , f_B and f_T to satisfy the inequality:

$\forall x \in V_{p_2}: \forall y \in V_{p_3}$:

$[(f_{IA}(f_A(x, y)).f_T(x, y) = y - \Delta V_A) \wedge (f_{IB}(f_B(x, y)).f_T(x, y) = y - \Delta V_B)]$.

Close in Mode


The mission controller is in this mode when the inlet valves are being closed. During this mode Plant select must be at start, the Indicator must be at green, ValveC must be closed, and the

Controller Specifications

regulator must be off. The mission controller must close ValveA and ValveB and then leave this mode. The mission controller must spend at most OT seconds in this mode.

Close in = $\langle \text{true}, Fv, p_{25} = 0 \wedge p_{26} = 0, 0, OT \rangle$.

Amber Mode

CF(Amber) = 

Indicator Mode

The mission controller is in this mode when the Indicator is being set to amber. During this mode, Plant select must be at start, the Indicator must be at green or amber, ValveA, ValveB and ValveC must be closed, and the regulator at off. The mission controller must turn the Indicator to amber, and then leave this mode. The mission controller must spend at most At seconds in this mode.

Indicator = $\langle \text{true}, p_5 = \text{start} \wedge p_{18} \in \{g, a\} \wedge Vc \wedge p_{28} = \text{off}, p_{18} = a, 0, At \rangle$.

Activate Mode

CF(Activate) = 

Reg on Mode

The mission controller is in this mode while the regulator is being set up. During this mode, Plant select is at start, ValveA, ValveB and ValveC are closed and the Indicator must be at amber. The mission controller must switch the regulator on and set it to the required temperature (i.e. the set point value), and then leave this mode. The mission controller must not spend more then Ro seconds in this mode.

Reg on = $\langle \text{true}, p_5 = \text{start} \wedge Vc \wedge p_{18} = a, Rset, 0, Ro \rangle$,

where Rset = $(p_{28} = \text{on} \wedge p_{29} = p_4)$.

Heat up Mode

The mission controller is in this mode while the contents of the vessel is being raised to the activation temperature. During this mode, Plant select must be at start, the Indicator must be at

Controller Specifications

amber or red, ValveA, ValveB and ValveC must be closed, and the regulator must be set. The mission controller must turn the Indicator to red, and then leave this mode. The mission controller must spend between $RT(p_4)$ and $RT(p_4) + \Delta ht$ seconds in the mode.


Heat up = $\langle \text{true}, p_5 = \text{start} \wedge p_{18} \in \{a, r\} \wedge \forall c \wedge Rset, p_{18} = r, RT(p_4), RT(p_4) + \Delta ht \rangle$.

Specification Condition

The upper bound of the controller graph for the activate mode must be less than (or equal to) the upper bound of the Activate mode.

So we must choose R_o and Δht to satisfy the inequality: $R_o + RT(p_4) + \Delta ht \leq f_{RT}(p_4)$.

Production Mode


CF(Production) = 

Reaction Mode

The mission controller is in this mode while the reaction is in progress. During this mode Plant select must be at start or collect, the regulator must be at red, ValveA, ValveB and ValveC must be closed and the regulator must be set. The system must leave this mode as soon as Plant select is at collect.

Reaction = $\langle \text{true}, p_5 \in \{\text{start}, \text{collect}\} \wedge p_{18} = r \wedge \forall c \wedge Rset, p_5 = \text{collect} \rangle$.

Collect Mode

CF(Collect) = 

Open out Mode

The mission controller is in this mode when ValveC is being opened. While the mission controller is in this mode the Plant select must be at collect, the Indicator must be at red, ValveA and ValveB must be closed and the regulator must be set. The mission controller must open ValveC, and then leave this mode. The mission controller must not spend more than O_c seconds in the mode.

Controller Specifications

Open out = $\langle \text{true}, p_5 = \text{collect} \wedge p_{18} = r \wedge IVc \wedge Rset, p_{27} = C_{\max}, 0, Oc \rangle$,

where $IVc = (p_{25} = 0 \wedge p_{26} = 0)$.

Empty Mode

The mission controller is in this mode while the product is being collected. During this mode, Plant select must be at collect, the Indicator must be at red, ValveA and ValveB must be closed and the regulator must be set. The mission controller must start to close ValveC and then leave this mode. The mission controller must spend between ETC and $ETC + \Delta et$ seconds in this mode.

Empty = $\langle \text{true}, p_5 = \text{collect} \wedge p_{18} = r \wedge IVc \wedge Rset, p_{27} < C_{\max}, ETC, ETC + \Delta et \rangle$.

Close out Mode

The mission controller is in this mode when ValveC is being closed. During this mode Plant select must be at collect, the Indicator must be at green or red and ValveA and ValveB must be closed. The mission controller must turn the Indicator to green, close ValveC, switch the regulator off, and then leave this mode. The mission controller must spend at most Co seconds in this mode.


Close out = $\langle \text{true}, p_5 = \text{collect} \wedge p_{18} \in \{g, r\} \wedge IVc, p_{27} = 0 \wedge p_{18} = g \wedge p_{28} = \text{off}, 0, Co \rangle$.

Specification Condition

The upper bound of the controller graph for the collect mode must be less than (or equal to) the upper bound of the collect mode.

So we must choose Oc , Δet and Co to satisfy the inequality: $Oc + ETC + \Delta et + Co \leq \Delta C$.

Operator Mode

CF(Operator) = 


Wait Mode

The mission controller is in this mode while the operator decides if another batch of C is to be produced or the (mission) system should be shutdown. During this mode, Plant select must at collect, off or on, the Indicator must be at green, ValveA, ValveB and ValveC must be closed and the regulator must be off. The mission controller must leave this mode as soon as the Plant

select is at off or on.

$\text{Wait} = \langle \text{true}, p_5 \in \{\text{collect}, \text{off}, \text{on}\} \wedge p_{18} = g \wedge Vc \wedge p_{28} = \text{off}, p_5 \in \{\text{off}, \text{on}\} \rangle.$

End Mode

$\text{CF}(\text{End}) =$ 

Turn off Mode

The mission controller enters this mode when no more C will be produced. At the start of the end mode the Plant select is at off. During this mode, the Indicator must be at green, ValveA, ValveB and ValveC must be closed, and the regulator must be off. The mission controller remains in this mode to the end of the system lifetime.

$\text{End} = \langle p_5 = \text{off}, p_{18} = g \wedge Vc \wedge p_{28} = \text{off}, \Omega \rangle.$

6.5. Summary and Conclusions

This chapter has discussed the formal constructs that will be used to express the descriptions and specifications produced during the controller analysis. The concepts introduced during the chapter were illustrated by examples from the reaction vessel (discussed in chapter two).

The *safety environment description*, is expressed as a history description. The *safety controller specification* is expressed as an SEMG. To structure the production of the safety controller specification a general structure is introduced, in terms of phases. There are five such phases: *start up*, *monitor*, *recovery*, *reset*, *shut down* and *end*. The general representation of each phase in the formal model is discussed.

The *mission environment description* is expressed as a history description. To describe the restrictions over the mission-oriented behaviour at the controller level it is necessary to specify relations that are imposed over all the state variables of a system. This introduces the possibility of imposing relationships in which the mission controller could influence the affect of the safety controller on the physical process. To prevent this, a restricted class of relations, called *dependent relations*, was defined. The *mission controller specification* is specified as an SEMG. A basic strategy for the development of the mission controller

specification was discussed in which an SEMG is constructed for each mode in the mission real world specification, was briefly discussed

Simply representing the behaviour of the safety controller in terms of the formal model will contribute little to the safety of the system. The analysis followed to produce the safety controller specification or safety controller description may be incomplete; or the formulation of these specification or description may be flawed (i.e., from simple errors in the notation, misunderstandings in the semantics).

For the formalism to make a significant (positive) contribution to the safety of the system, we must be able to provide a concrete argument for the assertion that the behaviour expressed by the safety controller specification is adequate to ensure that the safety real world specification will be maintained. Two ways for providing evidence for this argument are by: i) following a systematic step-by-step technique for the analysis of the safety controller and safety real world specification to produce the safety controller description and safety controller specification; and ii) performing a verification of the safety controller specification and safety controller description against the safety real world specification (the validation of the real world specification is discussed in the next chapter).

Similarly, simply representing the behaviour of the mission controller in terms of the formal model will contribute little to the reliability (in terms of the mission) of the system. As for the safety controller, a systematic analysis technique and verification technique should be used to provide evidence for the assertion that the mission controller specification is adequate for the mission real world specification.

Systematic analysis techniques and verification techniques for the specifications (and specification conditions) produced during the controller analysis are discussed in detail, in chapter 8.

Chapter 7 - Real World Analysis

7.1. Introduction

The formal structures for the representation of the specifications produced during the real world analysis, were discussed in chapter 5. In this chapter guidelines for the analysis of the system at the real world level are discussed in detail. To illustrate how these guidelines can help during the real world analysis of a particular system, a case study is presented in appendix B. The case study describes the real world analysis of a safety-critical chemical plant. During this chapter, simple examples from the reaction vessel are given to illustrate some parts of the methodology.

At the start of the real world analysis we have a system concept (see chapter 2). The system concept of a system is a model of the behaviour required from that system, presented as an informal specification, expressed in a suitable notation. As such the system concept suffers from the drawbacks of an informal specification, discussed in chapter 2.

The task of a real world analysis is to analyse the system concept in order to produce a formal requirements specification and to formally capture the restrictions imposed on system behaviour by the environment. Simply representing the system concept as a set of formal constructs would be of limited help. In fact, a messy and unstructured formal requirement specification would probably be less useful, as a description of system behaviour to the subsequent analysis of the system, than the system concept. Hence, a structured approach to the production of a structured set of formal requirements specifications from a system concept is necessary.

As a result of the real world analysis four formal specifications will be produced: the *safety real world description*, the *safety real world specification*, the *mission real world description* and the *mission real world specification*. Thereby, allowing a distinction between the restrictions imposed on system behaviour by the environment and the system behaviour required; and the safety-critical and mission-oriented behaviour of the system.

Most requirements engineering techniques have attempted to transform a system concept to a formal requirements specification – using iterative methods [Rzep85]. These

methods are usually based on an iterative process of analysing system behaviour as defined by the current specification (initially, this is the informal specification), documenting the requirements insights (thus producing a new specification), and checking the accuracy of the understanding (new specification) so gained.

The guidelines for the real world analysis will be presented as systematic techniques for the development of the four real world formal specifications. These systematic methods provide a set of step-by-step guidelines for the analysis needed to produce a formal specification; and a set of guidelines for the validation of the specification produced. Two reasons why the disciplined approaches (presented in this chapter) will improve confidence in the accuracy of the formal specifications are: i) the systematic analysis increases confidence in the completeness of the analysis and ii) the validation guidelines provide a means to systematically review the produced formal specifications.

The real world analysis starts with the construction of an initial real world description. After the initial analysis, the real world analysis is partitioned into *safety real world analysis* and *mission real world analysis*, these are performed in two main stages, the stages and the interactions between them are shown by figure 7.1. In this diagram, the boxes represent the stages, the downward arcs represent the order in which the analysis should be performed; and the upward arcs represent backtracking. The *hazard specification analysis* and mission real world specification analysis is performed in two main stages, the stages and the interactions between them are shown by figure 7.2. The main teams involved in the real world analysis are the customer, disaster analysts, hazard analysts and mission analysts.

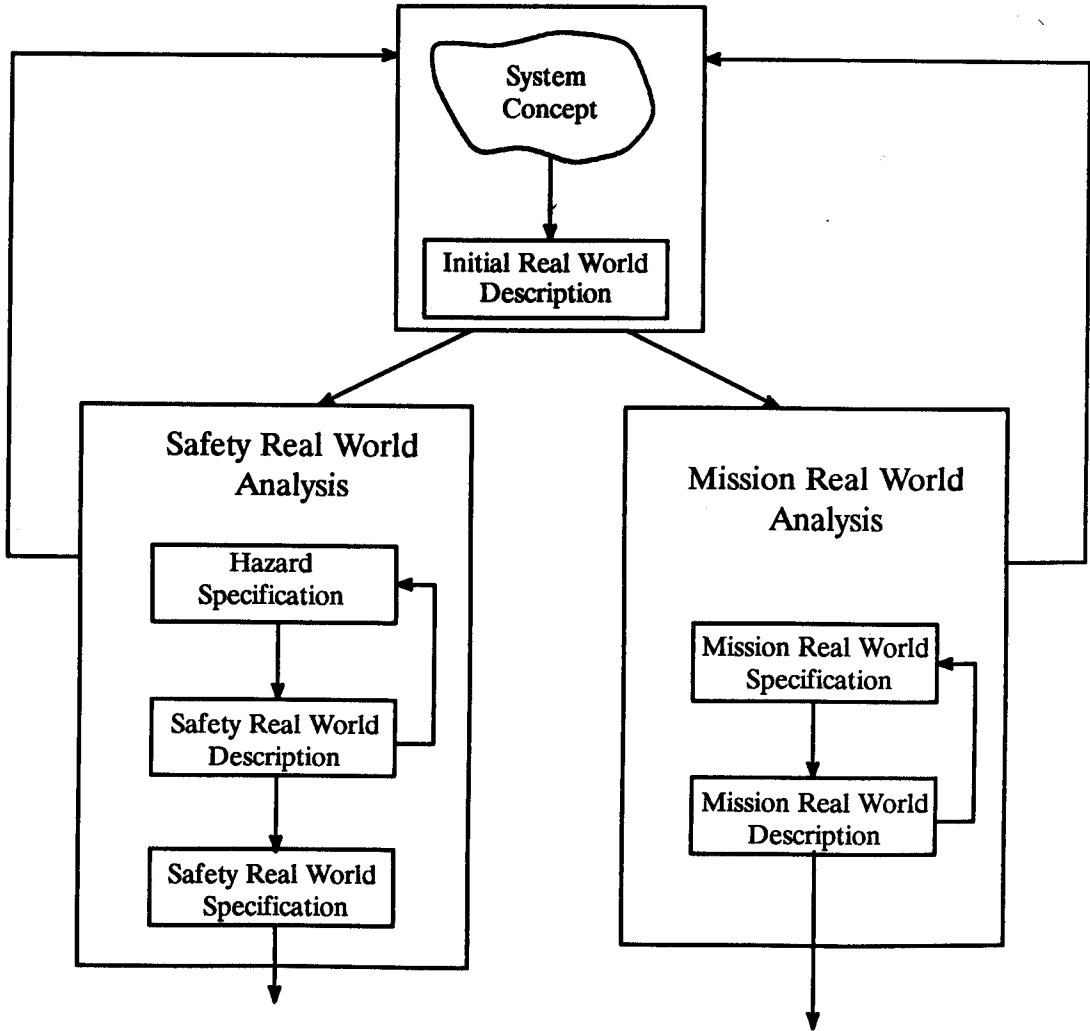


Figure 7.1. Real World Analysis

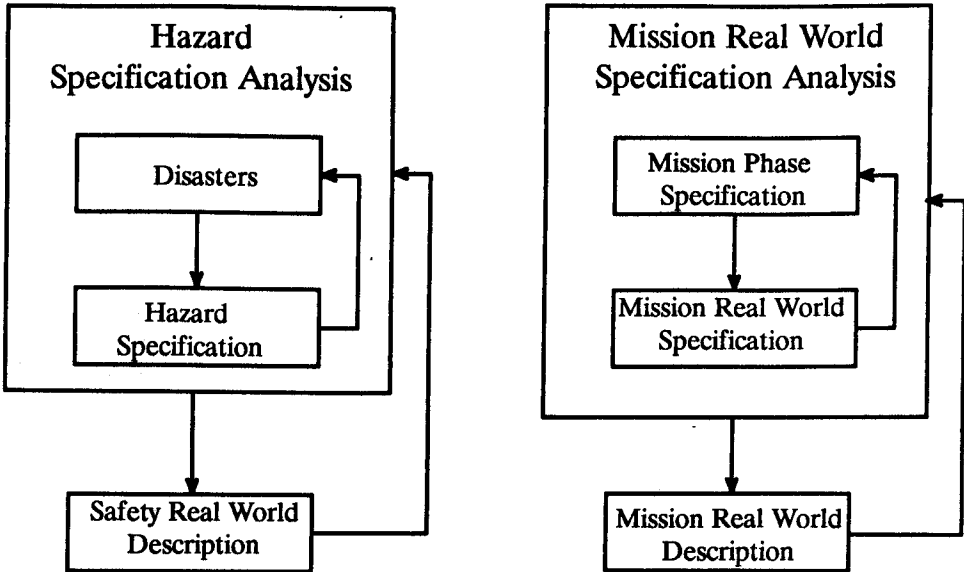


Figure 7.2. Real World Specification Analysis

7.2 Initial Real World Description Analysis

Before proceeding to a detailed analysis of the safety and mission issues of the system, a history description that specifies the real world variables of interest and the restrictions imposed on those variables by the environment should be produced. This history description will be called the initial real world description (denoted by IRWD). In this section a simple set of guidelines, are presented for the construction of an initial real world description. It should be recognised that the real world description produced at the end of the analysis will probably be incomplete (in the sense that some real world variables and relations will not be identified), but it should provide a useful basis for further analysis. This incompleteness will not affect the safety of the system, since a more thorough analysis will be performed later.

The formal structure of a history description ($\langle T, R_v, VP, CP, IR, HR \rangle$) can be used to derive guidelines for the production of the initial real world description.

7.2.1. Production Guidelines

The steps (given below) act as a simple guide to the disaster analysis team for the production of the IRWD from an analysis of the system concept, and knowledge of the application area.

Step 1

The *time base* is simply a representation of the lifetime of the system. In this elementary step, the main issues of concern to the analysis team would be the units, that is, seconds, milliseconds etc.

Step 2

A *real world variable* is a time varying quantity which represents a part of the state of the physical process or operator console. The task of the analysis team is to identify the real world variables of interest in describing system behaviour, the analysis proceeds in two stages.

a. All the real world variables that are mentioned in the system concept must be identified. This can be performed by identifying all the phrases that are used to describe a property of

the physical process, or a component of the operator console, in the text or tables of the system concept. Any diagrams used in the system concept should be inspected for any real world variables. For each of the identified variables the units must be determined.

The results of this analysis are recorded as a state variables table, that contains a row for each identified variable.

For example, let us suppose that performing the analysis indicated in *step 2* for the reaction vessel leads to the identification of a state variable which represent the volume of liquid in the vessel, with the units dm^3 . These results should be recorded as a row in the state variables table, for the reaction vessel this was recorded as: variable p_{13} , with the units dm^3 , the name *Volume*; and comments which state that it represents the total volume of liquid in the vessel.

Step 3

Variable range represents the set of possible values for a real world variable. The task of the analysis team is to identify the range of each identified real world variable. This can be achieved by investigating the restrictions imposed on the value of the variable by the environment or the construction of the plant. The results of the analysis performed in this step are recorded as a variable ranges table.

For example, let us suppose that performing the analysis indicated in step for the state variable *Volume*, identifies that the volume of liquid is represented by a real number, with a lower bound of zero representing no liquid in the vessel; and an upper bound of V_{max} representing the vessel being full of liquid. These results should be recorded as a row which contains the state variable (p_{13}) for the volume in the variables column, the set $\{x \in R \mid 0 \leq x \leq V_{max}\}$ in the ranges column, and comments which state that V_{max} represents the volume of the vessel.

Step 4

A *variable class* describes the restrictions imposed on a variable during the system lifetime. The task of the team is to check the standard classes to identify which describes the properties of the function, if none is appropriate the team must introduce a new class.

The results of the analysis performed in this step are recorded as a category and class table.

For example, let us suppose that performing the analysis indicated in this step for the state variable *Volume* leads to the result that volume is a continuous variable. This result should be recorded as row which contains the state variable (p_{13}) in the variables column, *Real World* in the category column, *continuous* in the class column and comments which state why it is a continuous variable.

Step 5

An *invariant relation* represents a relation between a set of real world variables which must hold for all the system states due to restrictions imposed by the environment on system behaviour. The task of the team is to identify all relevant relations and specify them. Invariant relations can be identified by systematically checking the real world variables, in two stages.

- a. Check all real world variables with the same units (or dimensions), for any relations. This check can be performed by partitioning the set of real world variables into groups with the same units. Then within each group we identify relations, which follow from the construction of the plant. If relations do exist between the identified groups, they will usually be simple relations which are obvious from the system concept.
- b. Check all real world variables with the same class, for any relations. This check can be performed in a similar way to that of step 5.a.
- c. Check physical laws which govern the behaviour of the real world variables, to see if there are any such laws which are invariant relations. This can be achieved by systematically identifying all physical laws (that are used in the application area) which involve a particular real world variable.

If some variables, mentioned in the physical laws (identified above), are not in the variable sequence of the IRWD the analysis team should investigate if those real world variables should be added to the variable sequence (i.e. are they of interest for the particular system). If some variables are added to the variable sequence then the ranges and classes of the variables must be identified as indicated by *steps 3 and 4*. If a physical law

can be expressed as an invariant relation the appropriate invariant relation should be added to the sequence of invariant relations. If a physical law cannot be expressed as an invariant law it should be recorded and marked as requiring further analysis (in *step 6*, the law should be expressed by a history relation).

d. Identify initial conditions. This can be performed by checking if there are any initial conditions which can be stated over the real world variables, by systematically checking each variable.

The results of this analysis must be recorded in a description relations table.

For example, let us suppose that as a result of the analysis in *step 5.a* for the reaction vessel the variables VolumeA (p_{10}), VolumeB (p_{11}), VolumeC (p_{12}) and Volume (p_{13}) are placed in the group for variables with the units dm^3 . Furthermore the analysis concludes that volumes of intermediate products of A and B can be regarded as C, hence at any time the volume of liquid in the vessel is the sum of the volumes of A, B and C. This result must be recorded as a row in the description table for which: column No. contains the number of the relation, Related Variables the state variables p_{10}, p_{11}, p_{12} and p_{13} , column Relationship the system predicate $p_{13} = p_{10} + p_{11} + p_{12}$; and Comments that the volume of liquid in the vessel is the sum of chemicals A, B and C.

Step 6

A *history relation* represents a relation between a set of real world variables which must hold for all intervals of the system lifetime for the set of histories given by the description. The task of the team is to identify all such history relations and specify them. History relations can be identified by systematically checking the real world variables, in four stages.

a. Check all real world variables with the same class, for history relations which follow from the construction of the plant.

b. Check derivatives. In this stage the analysis team should check if there are any relations over the derivatives of the real world variables. Common relations being upper and lower bounds on the first derivative of a real world variable. A relation involving the second derivative of a real world variable p_i (for which a state variable that represents its first

derivative is not defined) cannot be expressed directly by an history relation. However, such a relation can be expressed by a pair of history relations, firstly a new real world variable (say, p_j) must be introduced and a history relation that expresses the fact that p_j is the first derivative of p_i formulated (i.e. $p_{i,t} = p_{i,0} + \int p_{j(t)} dt$). Secondly, the relation involving the second derivative of p_i formulated in terms of p_j . Obviously, if a real world variable is introduced into the real world variable sequence of the initial description the range and class of the variable must be defined as indicated by *steps 3 and 4*.

c. Any physical laws that were marked as requiring further analysis in *step 5*, should be represented as history relations.

The analysis process of *steps 2, 5 and 6* continue until the analysis team believe that all the real world variables of interest (in *step 2*) all invariant relations (in *step 5*) and all history relations (in *step 6*) have been identified. The initial real world description is an intermediate specification, it is not essential that all the real world variables or relations be identified at this stage, a deeper analysis of the real world will be performed later. Hence no validation guidelines are provided. Nevertheless, the analysts should attempt to capture as many real world variables and relations as possible.

Step 7

The IRWD is defined as $IRWD(SY) = \langle T, Sv, VP, CP, IR, HR \rangle$, where

Sv is the sequence of real world variables identified in *step 2*;

VP is the ranges of Sv identified in *step 3*;

CP is the classes of Sv identified in *step 4*;

IR is the invariant relations identified in *step 5*; and

HR is the history relations identified in *step 6*.

A pictorial description of the initial real world description analysis guidelines is shown in figure 7.3. The two clouds represent the information available at the start of the analysis (i.e., the system concept and the application area knowledge). The large box represents the process of identifying and encoding the components of the initial real world description. The small boxes represent the guidelines for the construction of the components of the IRWD (e.g., the box labelled by the invariant relations represents the process of identifying

and encoding the invariant relations). The arcs represent the interactions between the processes.

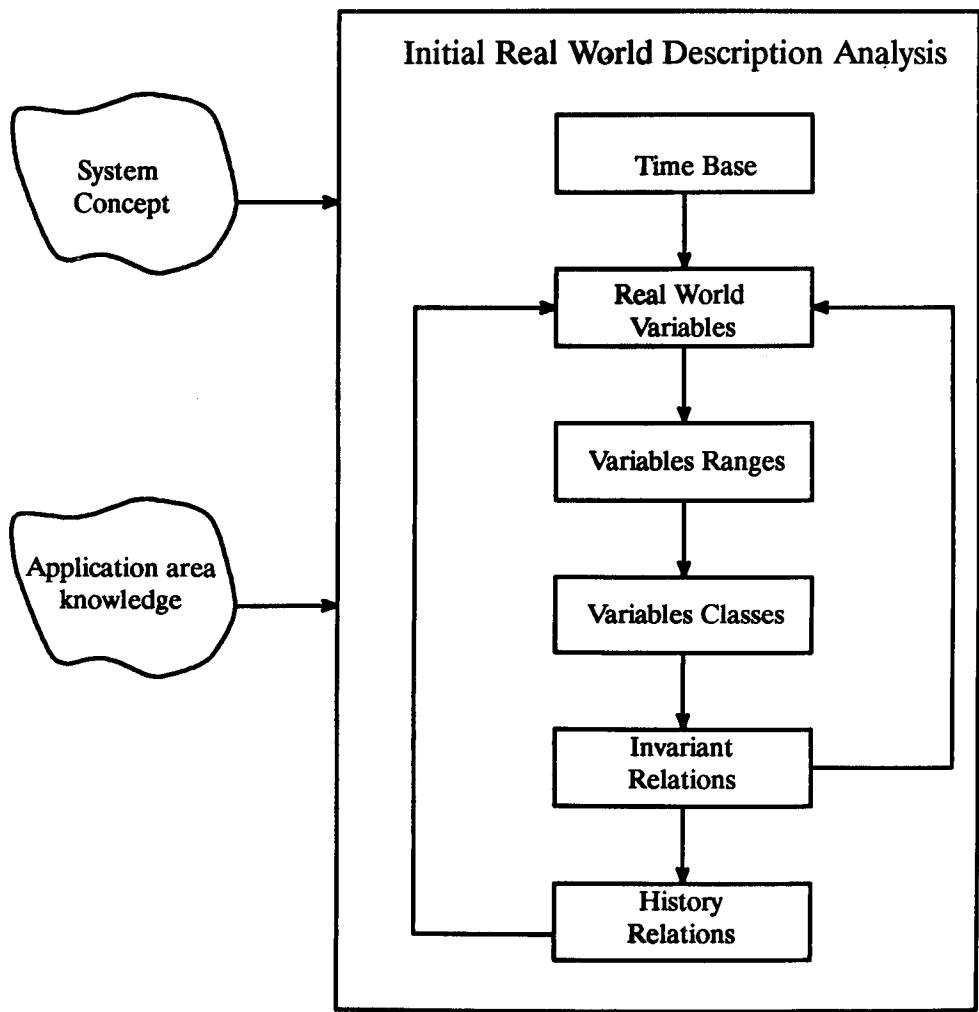


Figure 7.3 Initial Real World Description Analysis

7.3. Disaster Analysis

The strategies presented in this section are guides to the construction of the disaster set from the system concept and knowledge of what constitutes safety-critical behaviour in the application area (such as possible disasters in the application area and the risks associated with the materials used).

7.3.1 Disaster Identification

There are several traditional techniques which are used in the identification of disasters. These techniques are usually based on check-lists of possible disasters, either general or for particular types of physical process [Rodg71, Chem77].

Step 1

Choose (or construct) an appropriate check-list. A general check-list of the possible disasters associated with the application area of the system being produced should be available. If a check-list is not available then the disaster analysts would have to construct a check-list based on general check-lists; and the physical properties of the plant and its materials (e.g., for a chemical plant if a highly toxic material is being used, then release of such chemical may be a potential disaster). A typical check-list might include the following disasters:

1. Fire.
2. Explosion.
3. Uncontrolled toxic vapour or fluid release.
4. Personnel exposure to excessive heat.
5. Electrical shock to personnel.

The terms of the check-list should be defined as precisely as possible. For instance, the word “explosion” is sometimes used to describe incidents where there is just a loud noise. However, in the chemical industry the word properly describes incidents where there is a rapid release of energy with an accompanying blast wave capable of causing damage [Khar88]. To facilitate precise communication between the different members of the team,

agreement should be reached on the definition of the technical terms. Comprehensive and precise definition of terms which are commonly used are available [TRS83, ICE85].

Step 2

For each disaster on the check-list the disaster analysts must determine if the disaster is possible in the system or not. The results of the analysis should be recorded in a tabular format – as a disaster analysis table. A disaster analysis table has two columns; the headings and a brief description of the content of each column is given below.

- a. *Possible Disasters*: specifies the disaster of the check-list.
- b. *Results*: briefly summarise the results of the analysis on the disaster. If the disaster analysts believe that the disaster is not possible a brief justification should be given. If the disaster is recognised as a potential disaster, it should be represented in the formal model. Usually this can be achieved by simply introducing real world variables which can model the occurrence or non-occurrence of the disaster (see chapter 5).

For example, an extract of the disaster analysis table of the reaction vessel is given in table 7.1.

Table 7.1: Disaster Analysis of Reaction Vessel

Possible Disasters	Results
Toxic vapour or fluid release	The release of toxic vapours or fluid is not regarded as a potential disaster since the chemicals present (and their possible derivatives) in the system are non-toxic.
Explosion	An explosion is a potential disaster, the occurrence of which is denoted by P20.

Step 3

The state variables, that represent the disasters identified in *step 2*, must be described in a state variables table. The range of these variables will be the set {true, false}; and the class Catastrophe.

7.3.2. Validation Guidelines

It is very difficult to verify that all the potential disasters associated with a system have been identified. There are two possible causes for an incomplete disaster set: i) the disaster

analyst may reject potential disasters that are mentioned on the check-list; and ii) some potential disasters may not be mentioned on the check-list.

The approach suggested for the validation is based on two independent disaster analyses of the system. The approach is outlined below.

Step 1

Two disaster analysis teams should be formed (say, team A and team B). These team should perform a disaster analysis, to produce two disaster sets (SetA and SetB).

Step 2

Team A must check SetB by checking the justification given for rejecting possible disasters on the check-list of team B. Any possible additions to disaster set SetB must be noted. Similarly team B must check SetA.

Step 3

Any discrepancies between the analyses should then be resolved as follows:

- a.* Disasters included in SetA and SetB should be given common identifiers; these disasters represent an initial disaster set.
- b.* The remaining disasters of SetA, SetB and the additional potential disasters (produced in *step 3*) should be investigated; and those which are regarded as potential disasters by both teams added to the disaster set defined in *step 3.a*.

The disaster analysis task is completed when both teams agree that the check-lists used in the analysis have covered all possible disasters in the general application area of the system; and all potential disasters for the particular system under analysis, that are mentioned on a check-list, have been identified.

7.4. Hazard Specification Analysis

The hazard specification of a system is produced by identifying the hazards of the system, by a systematic analysis of the real world variables.

7.4.1 Hazard Identification

Step 1

In this step an analysis is performed over the real world variables, to determine those

variables that affect the disasters of the disaster set; and under what circumstances. This analysis proceeds by systematically answering the following questions for each state variable.

a. Can the variable influence the possibility of a disaster of the disaster set?

b. If the answer to *a* is yes, under what circumstances does the variable constitute a hazard?

The answer to question *b* may require the definition of additional real world variables, in which case the additional variables must be added to the real world variables under investigation.

If an analysis of the system concludes that for a particular state variable the answer to question *a* is no, a brief justification of why that variable cannot affect a disaster is required.

The results of this analysis should be recorded in a tabular format – as a *hazard analysis table*. A tabular format should be used, since it provides a structure for the recorded data; and by following the guidelines for the columns given below ensures that only essential information is recorded.

A hazard analysis table has three columns; the headings and a brief description of the content of each column is given below.

Variable: specifies the real world variable(s) under investigation.

Comments: informally describes the results of the analysis.

Hazard: System predicates that define the conditions under which the real variables under analysis constitute a hazard.

Step 2

In this step an analysis of the hazard analysis table constructed in *step 1* is performed to determine the hazard predicate of each disaster. This analysis is performed, for a particular disaster, by systematically checking the analysis table, to identify those system predicates that describe a hazard for that disaster. The disjunction of these system predicates defines the hazard predicate for that disaster.

Step 3

In this step the hazard specification of the system is constructed. The hazard specification is constructed by the disjunction of the hazard predicates for each disaster of the system.

Remark: The hazard specification of a system can, of course, be constructed directly from the table as the disjunction of the system predicates. However, introducing the intermediate *step 2*, improves the visibility of the link between the hazard specification and the disasters.

7.4.2. Validation Guidelines

There are three possible reasons why the hazard set of a system produced by following the guidelines in 7.4.1 may be incomplete, for a given set of disasters: i) that not all (relevant) real world variables have been checked, ii) that not all system conditions over the real world variables have been identified; and iii) that a mistake has occurred in the derivation of the hazard specification from the hazard analysis table. The validation scheme (like that of the disaster analysis) is based on an independent hazard analyses. The approach is outlined below.

Step 1

Two hazard analysis teams should be formed (say, team A and team B). These team should perform two independent hazard analyses; and produce two hazard analyses tables (TableA and TableB).

Step 2

Team A must check the analysis performed by team B, by checking TableB. Any differences, between the system predicates (Hazards) defined in TableA and TableB must be noted; and if team A identifies some additional potential hazards these must be recorded in a hazard analysis table. Similarly team B must check the analysis performed by team A.

Step 3

Any discrepancies between the analysis should then be resolved as follows:

a. Hazards included in TableA and TableB represent an initial combined table – TableC.

b. The remaining hazards of TableA, TableB and the additional potential hazards (identified in *step 2*) should be investigated; and those which are regarded as potential hazards by both teams added to TableC.

The advantage of performing the check over the hazard analysis table, rather than the hazard specification, is that it would be possible for two teams construct two different hazard analysis tables, but the same hazard specification. The significance of this is that by checking the the hazard analysis tables, we avoid the possibility of the hazard specification masking any difference in the analysis. Once a combined table (TableC) has been constructed the two teams can, of course, independently construct hazard specifications which can be compared.

7.4.3 Hazard Elimination

Once the hazards have been identified the hazard analysis team must decide if any of the hazards can be eliminated. That is, the first question the hazard analysts must ask is: “is their a safer process route?”. For example, if the hazard analysis of a chemical plant identifies a hazard caused by mixing two chemicals, which can lead to the release of toxic fumes. The hazard analysts must enquire if it possible to use different chemicals for which the hazard is not present. Since such solutions are clearly heavily application dependent, they are not treated in detail. However, as a final remark *elimination of the hazard*, should always be attempted before devices to minimize the possibility of a hazard are introduced to the (safety) controller.

If during the hazard analysis process, the hazard analysts identify some disasters which (they believe) were overlooked by the disaster analysts these should be recorded. The disaster analysts should consider these disasters. If as a result of a disaster analysis the disaster analysts agree that a disaster identified by the hazard analysts is possible, it must be added to the disaster set; and a hazard analysis performed for that disaster.

7.4.4. Complete Hazard Assumption

The hazards analysis task is completed when both teams agree that the complete hazard assumption holds for the system. Roughly speaking, the complete hazard assumption is

confirmed if the hazard analysts agree that for a disaster to occur during a history of the system at a time point t , the hazard specification must have been satisfied at a time point t' prior to t .

Three pieces of evidence for the argument that the hazard specification assumption holds, are provided from the systematic guidelines.

- a.* Complete disaster set. It can be argued that all the potential disasters have been identified, from the facts that an appropriate check-list was used to identify the disasters and two independent disaster analysis were performed.
- b.* It can be argued that all relevant safety real world variables have been identified from the facts that the SRD is constructed by an iterative process of checking all the real world variables against the hazard specification.
- c.* It can be argued that all the system conditions that can lead to a hazard have been identified from the fact that the hazard specification is constructed by a systematic and iterative process which ensures that all relevant variables have been considered.

7.5. Safety Real World Description Analysis

The safety real world description is constructed from the initial real world description; and the knowledge of the safety-critical behaviour gained from the hazard analysis.

7.5.1. Construction Guidelines

The steps (given below) act as a simple guide to the construction of the safety real description for the hazard analysts from the initial description and the hazard specification.

Step 1

The first task of the analysis team is to identify the safety real world variables of the system. This is performed in three elementary stages.

- a.* Define an initial set of safety real world variables as the variables over which the hazard specification is defined or those that represent the disasters of the system.
- b.* Identify all variables that are related to the above, by the relations of the IRWD. These can be identified by simply looking up the variables of the set defined in step *a*, in the related variables column of the relations table.

c. This step simply involves checking the ranges and classes of the variables identified in steps *a* and *b*, as defined by the IRWD.

Step 2

The task of the team is to identify those *invariant relations*, that impose restrictions on the safety-critical behaviour of the system. This task is performed in three steps.

a. Identify all relations over the safety real world variables in the IRWD. This elementary step can be performed by looking up the variables identified in *step 1* in the related variables column of the relations table.

b. Check if any relevant invariant relations were missed by during the IRWD analysis. This step can be performed by systematically investigating if the behaviour of a safety real world variable is influenced by any other real world variables, in the form of an invariant relation. This step may introduce new safety real world variables. If these variables are also new real world variables then the ranges and class of the variables must be defined.

The analysts have completed this step when all the safety real world variables have been checked for possible invariant relations.

Step 3

The task of the team is to identify those *history relations*, that impose restrictions on the safety-critical behaviour of the system. This task is performed in two stages, that are similar to those of *step 2*.

a. All history relations over the safety real world variables in the IRWD. This elementary step can be performed by looking up the variables identified in *step 1* in the related variables column of the relations table.

b. The analyst must systematically check all the safety real world variables to determine if there are any history relations involving them which have not been identified in *step 3.a*.

The analysts have completed this step when all the variables have been checked for possible history relations.

Step 4

The SRD is defined as $SRD(SY) = \langle T, Sv, VP, CP, IR, HR \rangle$, where *Sv* is the sequence of variables of IRWD and those identified in *step 1*;

- VP is the ranges of Sv;
- CP is the classes of Sv;
- IR is the invariant relations identified in *step 2*; and
- HR is the history relations identified in *step 3*.

A pictorial description of the safety real world description construction guidelines is shown in figure 7.4.

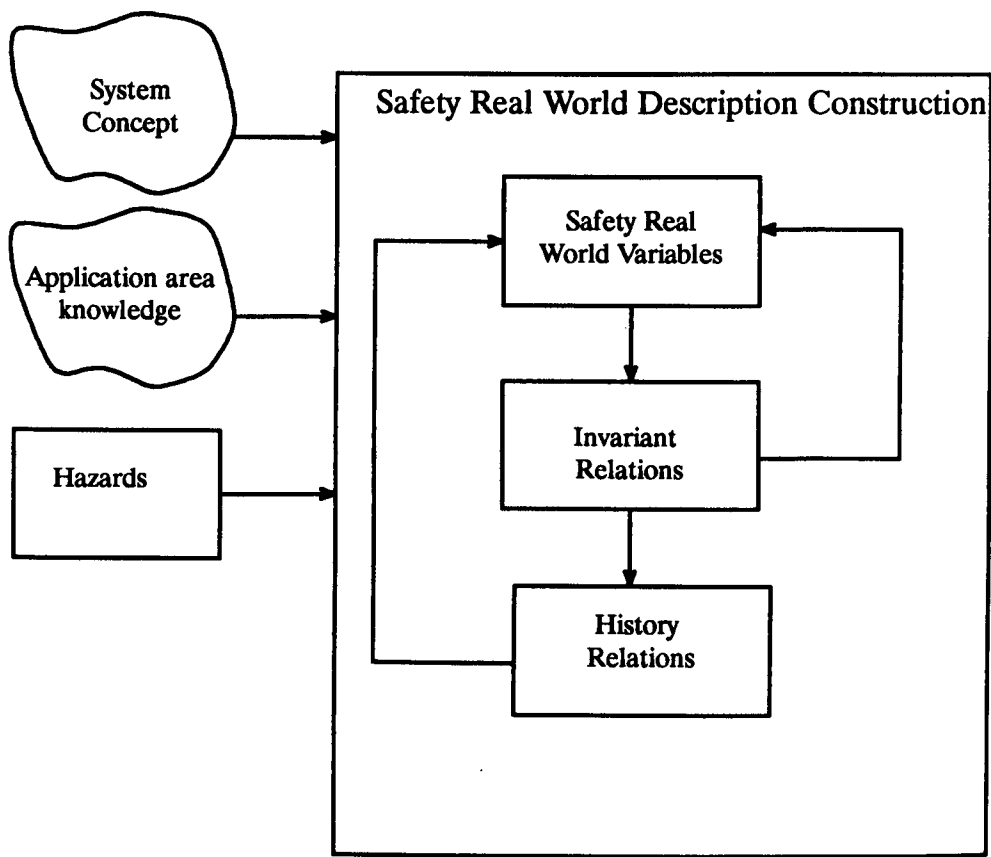


Figure 7.4 Safety Real World Description Construction

7.5.2. Validation Guidelines

A safety real world description, may be inadequate for three main reasons: i) some safety real world variables which are missing; ii) the ranges and classes of some variables do not capture the restrictions imposed on the variable accurately and iii) some relations may be missing or inaccurate representations of the restrictions imposed on system behaviour. A validation scheme is outlined below.

Step 1

Two analysis teams should be formed (say, team A and team B). These teams should perform safety real world description analysis as indicated in section 7.5.1; and produce two safety real world descriptions (DescA and DescB).

Step 2

Team A must check DescB, by inspecting the tables of DescA; and should note any ?. Similarly. TeamB must check DescA.

Step 3

Any discrepancies between the results of the analyses should then be resolved as follows:

- a. Common safety real world variables and relations of DescA and DescB form an initial safety real world description.
- b. The remaining safety real world variables and relations of DescA, DescB should be investigated; and those which are regarded as safety real world variables and relations by both teams added to the initial safety real world description.

This validation task is completed when both teams are satisfied that all safety real world variables have been identified and all relevant relations specified.

If new real world variables are introduced during the safety real world description analysis a hazard analysis must be performed over these new variables. If the revised hazard analysis in turn identifies any new variables the safety real world description analysis should be repeated for these new variables.

7.6. Safety Real World Specification Analysis

Step 1

An initial safety real world specification is simply produced as the negation of the hazard specification,

Step 2

The safety real world analysts must check that imposing the safety real world specification (produced in *step 1*) over the histories of the system will allow the mission to be fulfilled.

Step 3

For systems with complex safety real world specifications, the production and verification of the safety controller specification can become difficult. The safety analysts should investigate the relations of the SRD to determine if it is possible to define a simple safety real world specification that is stronger than the negation of the hazard specification. Of course, this new safety real world specification should also allow the mission to be fulfilled.

For most systems, the three steps given above should be straightforward. However, if necessary two safety real world specifications can be produced and compared.

7.7. Mission Real World Specification Analysis

The mission real world specification is produced by a systematic analysis of the system concept and IRWD. As was pointed out in chapter 4, a formal analysis should not be attempted until a preliminary informal (but structured) analysis has been performed. Hence the mission real world analysis is performed in two main stages. During the first stage the the mission phase specification is constructed; during the second stage the mission phase specification is analysed to produce the mission real world specification.

7.7.1. Mission Phase Specification Analysis

The structure of a phase graph ($\langle PH, A, S, E \rangle$) can be used to guide the analysis. The strategy given in this section guides the construction of the mission phase specification from an analysis of the system concept, safety real world specification and initial real world description. At the end of the mission phase specification analysis, the basic structure of the system (i.e. the relationship between the main tasks of the system) should be reflected by the mission phase specification; and the system concept should be structured into a set of documents such that each document is related to a phase. The guide to the production of the mission phase specification is presented in three stages: *high-level phase analysis*, *phase analysis* and *phase check*.

High-Level Phase Analysis

In this stage a high-level phase graph is produced consisting of three phases: *start phase*, *mission phase* and *end phase*. The main purpose of this stage is to structure the system concept into three documents related to the start up of the mission subsystem, the mission to be achieved by the mission subsystem and the shut down of the mission subsystem.

Step 1

The *start phase* is an informal specification of the start up procedure of the mission subsystem. The mission analyst must specify the start phase by giving: i) an informal description of the behaviour at the start of the system; ii) informal description of the behaviour of the system during the start phase; iii) informal descriptions of the events which specify the completion of the phase and iv) any constraints imposed on the duration of the start phase. In the informal descriptions any real world variables are referred to by the name given in the variables table.

Those parts of the system concept that are related to the start phase must then be identified; and recorded as a separate document. This can be achieved by checking the informal specification of the start phase against the system concept to identify related requirements. These requirements are referred to as the phase requirements of the start up phase.

Step 2

The *mission phase* is an informal specification of the mission to be performed by the system. The mission analyst must specify the mission phase in the same format as that of the start phase. Then identify and record the phase requirements of the mission phase.

Step 3

The *end phase* is an informal specification of the shut down procedure of the mission subsystem. The mission analyst must specify the end phase in the same format of the start phase. Then identify and record the phase requirements of the end phase.

The (trivial) phase graph produced by the high-level graph analysis is shown in figure 7.5.



Figure 7.5. High-Level Phase Graph

Phase Analysis

In this stage the high-level phases are analysed, by checking their requirements and the phase specifications, to identify a phase graph that decomposes the requirements of the phase. To keep the complexity of the analysis low, the phase sequences of the graphs should be kept small (up to five phases). During this analysis the mission analysts must maintain a formal representation of the structure of the current phase specification. The analysts must select a phase (initially this will be the start phase); and then proceed with the following five steps.

Step 1

Identify a phase graph that specifies a set of possible behaviours during the selected phase. This can be defined by identifying the tasks that must be performed during the selected phase for the requirements of the phase to be fulfilled. These tasks are then represented as phases; and connected to perform a phase graph that specifies a set of phase sequences that perform the task defined by the selected phase. The analysis team must clearly identify the requirements of the selected phase that are captured by the defined phase graph.

Step 2

The analysis team must modify the phase graph, produced by the analysis in *step 1*, to capture any requirements of the selected phases. This step should be repeated until the analysis team are satisfied that all the requirements of the selected phase have been captured, by the current phase graph.

Step 3

The phase graph formed by the decomposition of the selected phase is linked to the mission phase specification (from which the phase was selected) by adding arcs from all the

predecessors of the selected phase to the start phases of the phase graph; and adding arcs from the end phases of the phase graph produced to the successors of the selected phase.

Step 4

The detailed analysis of the phases may highlight some problems in the requirements of the phases, such as ambiguities and inconsistencies. These problems should be noted and discussed with the customer before the analysis proceeds. Any changes, to the requirements of the phase should be reflected in the phase graph.

Step 5

The phase requirements of the selected phase should be structured in accordance to the phases, produced in steps 1 to 3.

After phase graphs have been produced for the high-level phases, the mission analysts must check the new phases to identify those phases which require further analysis. The identified phases are then analysed by following the guidelines given in steps 1 to 5. This is then repeated until the phases no longer require further analysis.

There are no strict rules for when a phase no longer requires further analysis, the mission analysts must use their judgement to decide when a portion of the system concept has been sufficiently structured to allow a detailed formal analysis to proceed. General speaking, a phase does not require further analysis when its behaviour can be represented by a closely related set of tasks (as a rough guide line, when the associated part of the system concept is one page). Hence the mission phase specification analysis is complete when the mission analysts believe that the phases of the mission phase specification no longer require further analysis.

In essence, the phase analysis process produces a sequence of mission phase specifications starting from the high-level phase specification to the final mission phase specification. If the final mission phase specification is a complex graph (i.e., with a large number of phase sequences) the mission analysts should record an intermediate mission phase specification (IMPS) and the relationship between the phases of the intermediate phase specification and the final phase specification. This relationship is recorded as a function which maps the phases of the IMPS to a phase graph that forms part of the final

mission phase specification. A suitable intermediate mission phase specification, would be one that consisted of a small number of phase sequences (say, about twenty); and for which each phase is represented by a phase graph with a small number of phase sequences, in the final mission phase specification. Such an intermediate mission phase specification will be used check the final mission phase specification.

Mission Phase Specification Check

Any ambiguities or inconsistencies highlighted during the production of the mission phase specification are checked by the customer (step 4). However, a check should still be performed over the completed mission phase specification. A simple two step check is presented.

Step 1

Check that the developed graph is a single entry exit phase graph. For many mission phase specifications this check can be confirmed by simply inspecting the graph. If the graph is too complicated (for a visual inspection) then the formal representation should be checked.

Step 2

Check that behaviour expressed by the phase graph captures the intentions behind the system concept. The two main causes for a phase graph being an inaccurate representation of the mission requirements of the system concept are: i) some phase sequences that comply with the system concept are not defined by the mission phase specification; and ii) some phases that are defined by the mission phase specification do not comply with the system concept.

A mission phase specification (MPS) is checked in three stages.

a. If the MPS defines a small number of phase sequences, then the analysts proceed directly to step 2.*b.* However, if the MPS defines a large number of phase sequences the IMPS should be presented to the customer to check if the customer agrees with the identification of the main phases of system behaviour. Once the IMPS has been confirmed the mission analysts proceed with *steps 2.b* and *2.c* for the phase graph of each phase of the IMPS.

b. The structure of the phase graph should be presented to the customer. The consequences of the structure are presented by discussing the phase sequences defined by the graph. However, if the customer disagrees with the inclusion of a phase sequence, or the exclusion of a phase sequence the discrepancy should be noted.

If the discrepancy is caused by the inclusion of a phase sequence then the mission analysts should justify the inclusion of the phase sequence by referring to the system concept. However, if the customer disagrees with this justification then the phase graph should be modified by removing (or modifying) the phase sequence and (if necessary) the system concept modified to remove the cause of the discrepancy.

If the discrepancy is caused by the exclusion of a phase sequence then the customer should indicate which requirements are not captured by the phase sequence; and if possible how they can be captured. If the mission analysts believe that requirements mentioned by the customer are captured elsewhere they should explain how they are captured. If the customer is satisfied with the explanation of the mission analysts then the check can continue. However, if the customer is not then a phase sequence which captures the requirements should be included; and the cause of the discrepancy in the system concept removed.

c. The phase specifications and the requirements of the phases should be presented to the customer. Basically, this is used to check that the customers and the mission analysts view of the role of the task represented by the phase is in agreement. The check is performed by the mission analyst presenting the phase and ensuring that there is general agreement over the phase specifications.

7.7.2. Mission Real World Specification Analysis

To produce the mission real world specification, the phases of the mission phase specification are replaced by modes (or SEMGs), which are then combined to develop the mission real world specification of the system.

A function can be defined from the structure of a mission phase specification to the SEMGs that will be constructed for its mission real world specification, this function is

called a behaviour function (denoted by BF).

Definition: Behaviour function

The behaviour function (BF) of a mission phase specification is a function from its phase set to a set of SEMGs, such that for a phase p $BF(p)$ returns the SEMG that formally expresses the behaviour expressed by p ; and for which the mode sets of the mode graphs given by the behaviour function are disjoint.

Behaviour Analysis

A two step guide to the construction of a behaviour function, by a formal analysis of the behaviour that should be exhibited by a phase, is presented. The following two steps must be performed for all the phases of the mission phase specification.

Step 1

The analysts must decide whether the behaviour specified by the phase and its associated requirements can be represented by a mode imposed over the real world variables, or whether an SEMG is required. A mode should be used if the only events of interest during the phase are the start and end event of the phase. This can be determined by systematically checking the phase requirements against the real world variables in order to identify any events of interest.

- a.* If a mode can represent the behaviour of the phase then the mode is constructed by an analysis of the phase as outlined in *step 2*.
- b.* If an SEMG is needed to represent the behaviour of the phase then a suitable mode graph must be sketched. This sketch should specify the interactions between the modes that are required in the formal specification of the phase, and provide informal specifications of the modes.

Step 2

For convenience, the structure of a mode is recalled: $\text{Mode} = \langle \text{Start}, \text{Inv}, \text{End}, \text{LB}, \text{UB} \rangle$.

- a.* The *start predicate* of a mode is constructed by systematically checking the informal start condition against each real world variable to determine the constraints imposed on the real world variables at the start of the mode.

- b. The invariant predicate* of a mode is constructed by systematically checking the informal invariant condition against each real world variable to determine the constraints imposed on the real world variables during the mode
- c. The end predicate* of a mode is constructed by systematically checking the informal end condition against each real world variable to determine the constraints which must be imposed on the real world variables for the task defined by the mode to be completed.
- d. The time bounds* of a mode are constructed by checking the informal specifications to determine the lower and upper bounds of the mode. This is performed in three steps.
 - i.* The analysts must determine if any timing constraints are required. Two broad classes of timing constraints are recognised. The first relates to timing constraints which are related to the continuity of the mission. For example, a task that is completed when a vessel contains a specified amount of chemicals, requires an upper bound to ensure that the mission controller will ensure that the task is completed once it has started. The second class relates to the behaviour that will be observed in the physical process. For example, a task that must maintain the temperature of the system within a specified range until a reaction is completed, requires a lower bound to ensure that the mission controller will maintain the temperature long enough for the reaction to be completed.
 - ii.* If constraints are required the analysts must determine if they are time values or time-valued functions. Unless it is obvious that a time-value is required, the analysts must systematically check the real world variables (in particular those variable that represent part of the state of the operator console), to see if they can affect the timing constraints.
 - iii.* Specify the appropriate value or functions. If a function is required then the relationship between the variables identified in *ii* and the timing constraint must be investigated.
- e. Informal specification.* Once the formal specification of a mode has been constructed an informal specification which describes the behaviour of the mode should be stated.

Behaviour Function Connection

The SEMGs given by a behaviour function are connected to form an SEMG by the function SEM.

Algorithm 7.1

Function SEM(PG: PhaseGraph, BF: BehaviourFunction): SEMGraph;

Var MG: SEMGraph;

IA, EA: SetOfModes;

m, w, x: Mode;

y, z: Phase;

1: $M(MG) := \bigcup_{m \in M(PG)} M(BF(m))$;

2: $IA := \bigcup_{m \in M(PG)} A(BF(m))$;

3: $EA := \{(w, x) \in M(MG)^2 \mid \exists (y, z) \in A(PG) \wedge w = E(BF(y)) \wedge x = S(BF(z))\}$;

4: $A(MG) := IA \cup EA$;

5: $S(MG) := S(BF(S(PG)))$;

6: $E(MG) := E(BF(E(PG)))$;

7: $SEMG := MG$;

8: **Stop**.

Comments

1: The *mode set* of MG is the union of the mode sets of the mode graphs given by the behaviour function.

2: The *internal arc set* of MG (i.e. the arcs of the graphs given by the behaviour functions) is the union of the arc sets of the mode graphs given by the behaviour function.

3: The *external arc set* of MG (i.e. the arcs which are used to connect the graphs given by the behaviour functions) is constructed as the set of all pairs of modes (w, x) for which there is an arc (y, z) in the phase graph such that the end mode of the behaviour graph of w is y ; and the start mode of the behaviour graph of z is x .

4: The *arc set* of MG is the union of the *internal* and *external arc sets*.

5: The *start mode* of MG is the start mode of the behaviour graph of the start phase.

6: The *end mode* of MG is the end mode of the behaviour graph of the end phase.

A pictorial description of the mission real world specification construction guidelines is shown in figure 7.6.

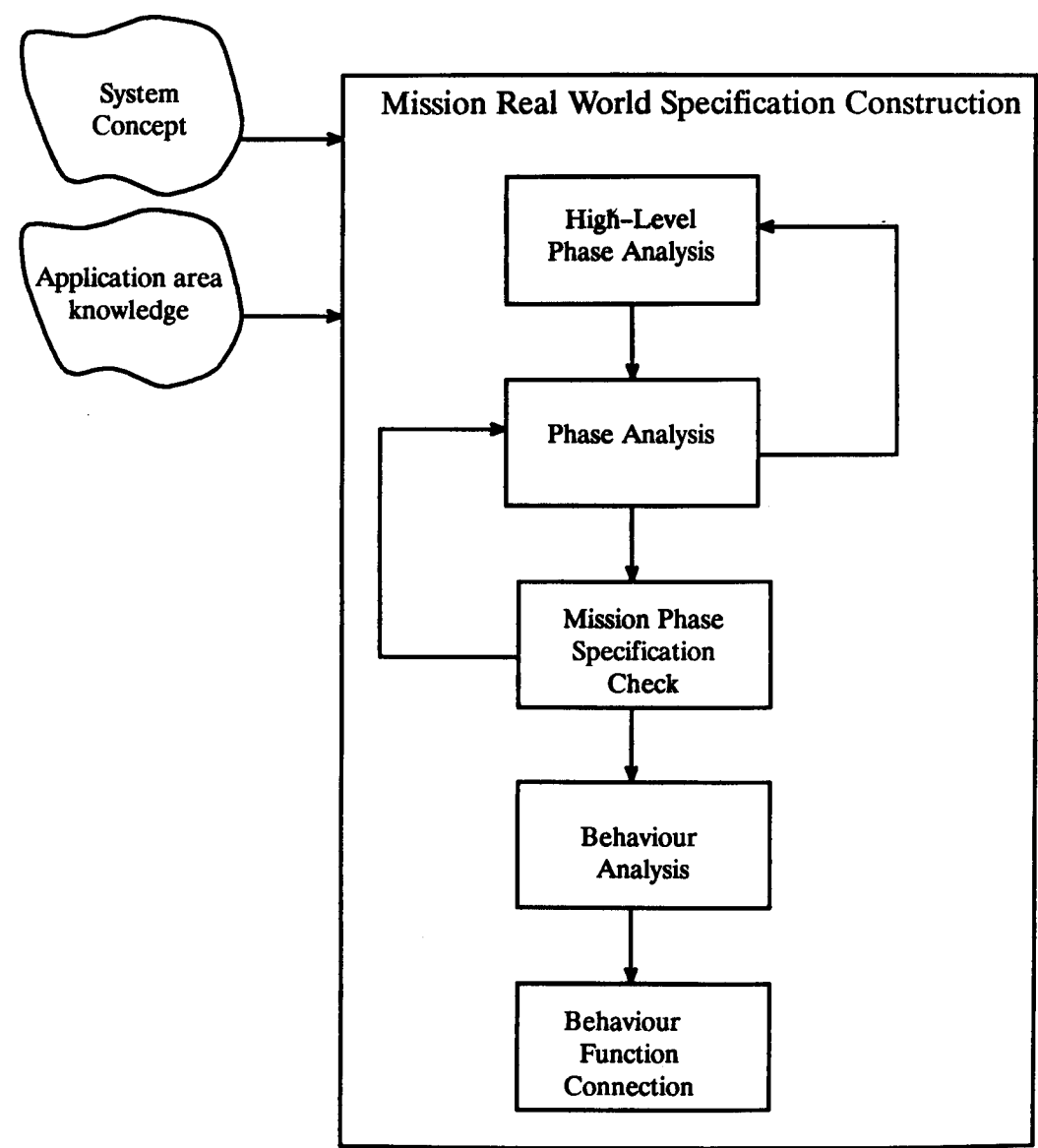


Figure 7.6. Mission Real World Specification Construction

7.8. Mission Real World Description Analysis

The mission real world description is constructed from the initial real world description; and the knowledge of the mission-oriented behaviour gained from the mission phase specification and mission real world specification analysis.

7.8.1. Construction Guidelines

The steps (given below) act as a simple guide to the construction of the mission real description for the mission analysts from the initial real world description and the mission real world specification.

Step 1

Identify the mission real world variables of the system. This is performed in three elementary steps.

- a.* Identify, all the variables over which the mission real world specification is defined.
- b.* Identify all variables that are related to the above, by the relations of the IRWD. These can be identified by simply looking up the variables identified in the related variables column of the relations table.
- c.* Check the ranges and classes of the variables, as defined by the IRWD.

Step 2

Identify those *invariant relations*, that impose restrictions on the mission-oriented behaviour of the system. This is performed in two steps.

- a.* Identify all relations over the mission real world variables in the IRWD or SRD. This elementary step can be performed by looking up the variables identified in *step 1* in the related variables column of the relations table.
- b.* The analyst must systematically check all the mission real world variables to see if there are any invariant relations involving them which have not been identified.

Step 2 is completed when all the variables have been checked for possible invariant relations.

Step 3

Identify those *history relations*, that impose restrictions on the mission oriented behaviour of the system. This is performed in two steps.

- a.* All history relations over the mission real world variables in the IRWD. This elementary step can be performed by looking up the variables identified in *step 1* in the related variables column of the relations table.
- b.* Systematically check all the mission real world variables to determine if there are any history relations involving them which have not been identified.

Step 3 is completed when all the variables have been checked for possible history relations.

Step 4

The MRD is defined as $MRD(SY) = \langle T, Sv, VP, CP, IR, HR \rangle$, where
Sv is the sequence of variables of SRD and those identified in *step 1*;
VP is the ranges of Sv;
CP is the classes of Sv;
IR is the invariant relations identified in *step 2*; and
HR is the history relations identified in *step 3*.

A pictorial description of the mission real world description construction guidelines is shown in figure 7.7.

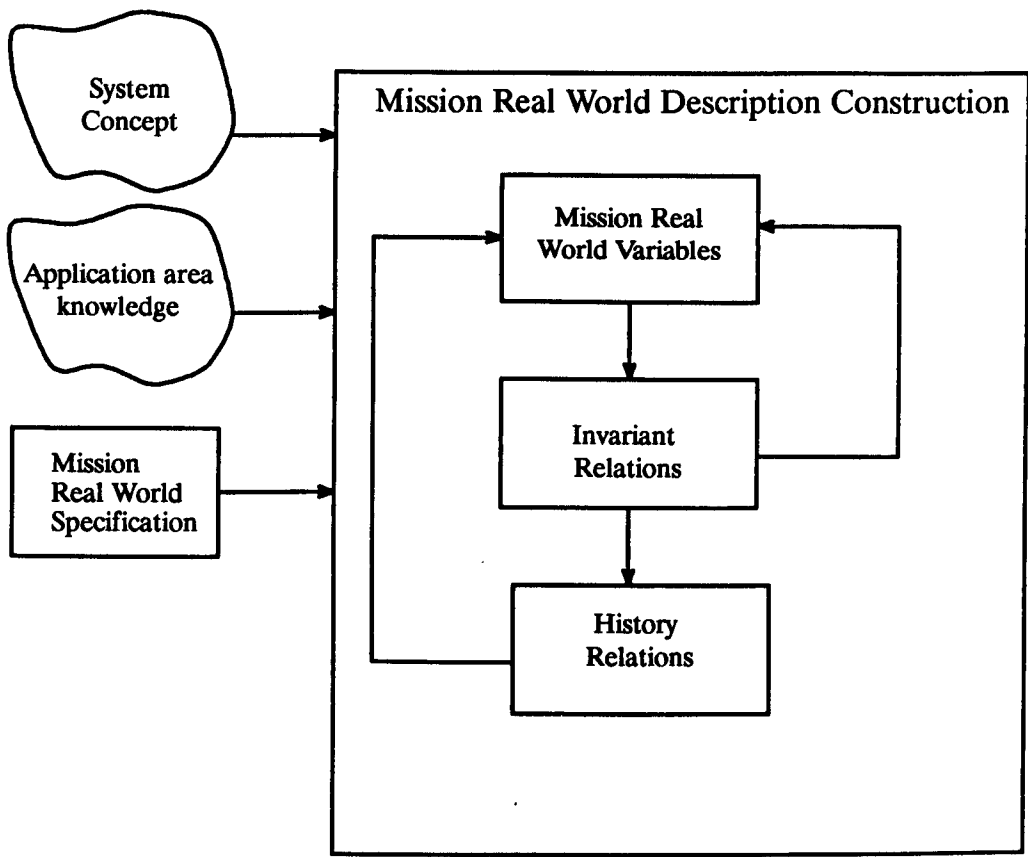


Figure 7.7. Mission Real World Description Construction

7.8.2. Mission Real World Specification Checks

After the mission real world description of a system is constructed, the mission analysts must confirm the completeness and consistency of the mission real world specification of

that system against this description. The checks which must be performed are given in chapter 4, for completeness see section 4.3.3., and for consistency see section 4.3.4.

7.8.3. Mission Validation

Mission validation is concerned with the validation of the mission real world specification. The validation must provide some evidence that the mission real world specification accurately reflects the intentions of the system concept. Recall that the mission phase specification has been checked, hence at this stage the validation should concentrate on the formalization of the phase of the mission phase specification. Before the validation proceeds, a cross check between the informal and formal specification of each mode should be performed, by the mission analysts. Even if it is confirmed that the mission real world specification has accurately captured the system concept, the customer may still requires changes due to omissions in the system concept that are highlighted by the increased understanding gained by checking the specification.

Step 1

The detailed behaviour of a phase as expressed by the behaviour graph is assessed. The assessment is performed by presenting the mode sequences of the behaviour graph to the customer and demonstrating how the mode sequences reflect the behaviour of the phase. The demonstration is performed by showing the following, for each phase sequence.

- a.* The start predicate of the start mode of the sequence ensures that the informal start condition of the phase is satisfied, at the start of the sequence.
- b.* The invariant predicate of the sequence ensures that the informal invariant condition of the phase is satisfied, during the sequence.
- c.* The invariant predicate of the sequence and the end predicate of the end mode ensure that the informal end condition of the phase is satisfied, only at the end of the sequence.
- d.* The timing constraints over the sequence ensure that the timing constraints over the phase are satisfied.
- e.* The sequence is in compliance with the phases requirements. This can be demonstrated by performing a dry run through the mode sequence; and showing how each mode and its

position in the mode sequence is derived from the phases requirements. Any discrepancies identified during *step 1*, should be clearly recorded.

Step 2

This step is entered for a phase, if the customer identifies any discrepancies between the requirements of a phase and the behaviour expressed by the behaviour graph of that phase. The customer should point out the cause of the discrepancy, e.g., the customer may say it was due to an oversight in the system concept. These discrepancies should then be resolved by modifying the phase requirements, behaviour graph or both.

A mission real world specification passes the validation stage when the customer and mission analysts are convinced (and can provide some evidence) that the mission real world specification accurately represents the intentions of the system concept.

7.9. Combination of Analysis

The real world analysis has considered the analysis of the safety and mission behaviour in isolation. Here the best way to combine the analysis is considered. It is suggested that the safety real world analysis should be performed before the mission real world analysis. The main advantage of performing the safety real world analysis first, is that the identification of potential disasters and hazards may lead to changes in the system concept, in some cases it may even stop development of the system. However, a drawback of performing the safety real world analysis first is that any changes made to the system concept must be reconsidered. Hence, the disaster table and hazard table must be checked after mission real world analysis. An example of how the mission real world specification of a system can be compared against the safety real world specification, is given in appendix B.

7.10. Summary

This chapter has introduced some guidelines for the development of the specifications produced during real world analysis. The real world analysis was presented in three main stages, firstly a preliminary analysis stage which leads to the production of an initial real world description, secondly a safety real world analysis which leads to the production of the

safety real world specification and safety real world description; and thirdly the mission real world analysis which leads to the production of the mission real world specification and mission real world description.

A strategy for the safety real world analysis was presented in four main stages: disaster analysis, hazard analysis, safety real world description analysis and safety real world specification analysis.

The strategy for the identification of the disasters, was based on constructing an appropriate check-list of disasters for the system, then systematically analysing the check-list and system to identify the potential disasters of the system. A systematic strategy for the identification of the hazards based on the knowledge of the disasters, was presented. A set of systematic guidelines for the production of the safety real world description from the initial real world description; and the knowledge of the safety-critical behaviour gained from the hazard analysis. Finally, a strategy for the production of the safety real world specification was presented. It was suggested that the specifications produced during the safety real world analysis can and should be validated, by two independent teams performing the analysis at each stage and comparing their results.

The strategy for the production of the mission real world specification was given in two stages: mission phase specification analysis and mission real world specification analysis. The strategy for the mission phase specification analysis was given in three stages, in the first stage a high-level analysis of the system concept is performed to produce a high-level phase graph. In the second stage, the phase specification is continually decomposed until the analysts believe it is sufficiently structured. In the third stage the mission phase specification is checked. The strategy for the mission phase specification is presented in two stages. In the first stage an SEMG is constructed for each phase, this is represented as a function (the behaviour function) from the phase set of the mission phase specification to a set of SEMGs. In the second stage, the SEMGs specified by the behaviour function are connected by applying the function SEM over the mission phase specification and the behaviour function.

A set of systematic guidelines for the production of the mission real world description from the initial real world description, and the knowledge of the mission-oriented behaviour gained from the mission real world specification analysis, were discussed.

Guidelines for the verification of the completeness and consistency of the mission real world specification and validation of the mission real world specification against the system concept were also outlined. These strategies exploit the information made available by the modular production of the specifications.

The separation of safety and mission issues simplifies the guidelines of both the safety and mission specifications. The safety analysis is simplified by focusing only on the potential disasters and their hazards; and the mission analysis is simplified by removing the necessity to consider the safety-critical issues in detail with the mission. The guidelines provide a framework in which the certification body can check the process used to develop the specifications. In particular, the guidelines for safety real world analysis enable the analysts to record development information, that can be used to support the validation of the specifications produced during the safety real world analysis.

Chapter 8 - Controller Analysis

8.1. Introduction

In chapter 6 the formal structures for the representation of the specifications produced during the controller analysis were discussed. In this chapter, methodologies to guide the controller analysis are discussed in detail. These methodologies will be presented as a set of systematic step-by-step strategies (guidelines). The case study of a safety-critical chemical plant will be used to illustrate the methodologies (the case study is presented in appendix C, the real world analysis for this plant is presented in appendix B).

At the start of the controller analysis we have four formal constructs which express system behaviour at the real world: the safety real world description, the safety real world specification, the mission real world description and the mission real world specification. As a result of the controller analysis four formal constructs will be produced. Two of these formal constructs express the relationship between system behaviour at the real world and controller levels: the safety environment description and mission environment description; the other two constructs express the system behaviour required at the controller level: the safety controller specification and the mission controller specification.

The controller analysis is partitioned into two distinct processes *safety controller analysis* and *mission controller analysis*.

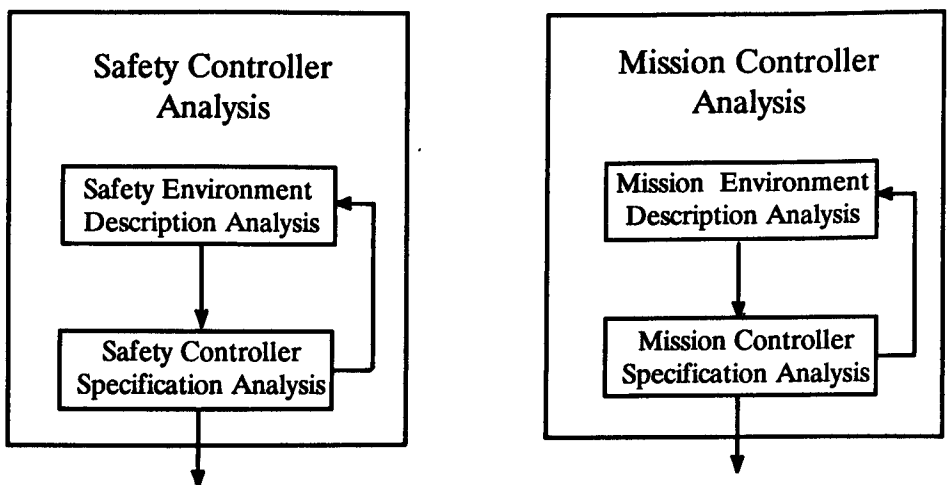


Figure 8.1. Controller Analysis

These are performed in two main stages, the stages and the interactions between them are shown by in figure 8.1. In this diagram the boxes represent the stages, the downward arcs represent the order in which the analysis should be performed; and the upward arcs represent backtracking between the stages due to violations in checks involving the produced specifications. The main teams involved during the controller analysis are the safety controller analysts and mission controller analysts. An overview of the safety controller analysis, is presented in the following paragraphs.

This chapter presents a set of *guidelines*, for the production of the safety environment description, based on an analysis of the properties of the actuators and sensors of the safety controller. Then a set of *guidelines*, for the production of the safety controller specifications based on a systematic analysis of the real world specifications and safety environment description of the system, are presented.

The situation for the mission controller analysis is the same as that for the safety controller analysis described by the paragraph above (just replace safety by mission). The safety controller specification analysis is performed in two main stages and the mission controller specification analysis in three main stages, the stages are illustrated in figure 8.2.

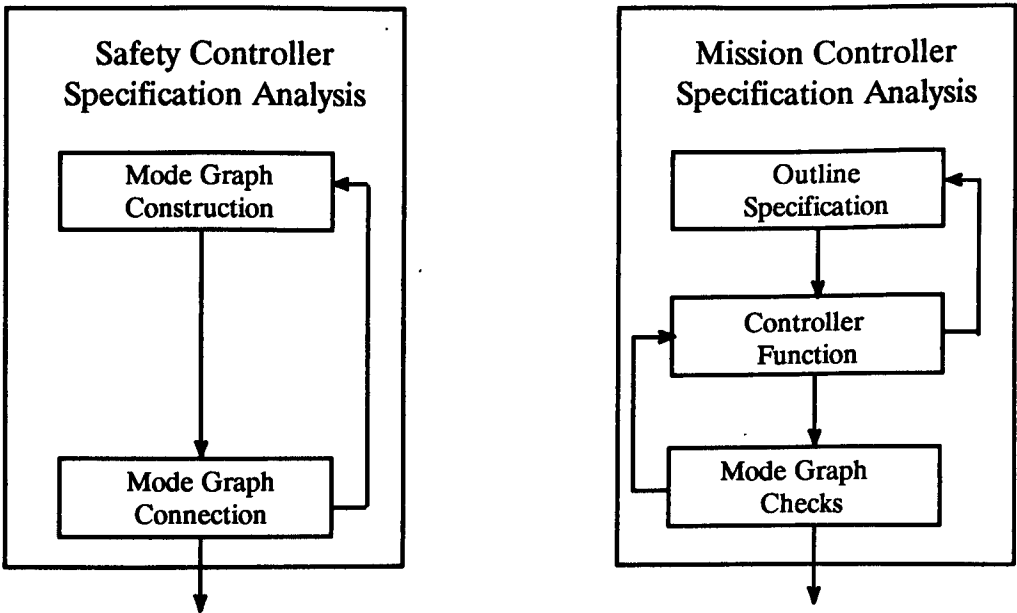


Figure 8.2. Controller Specification Analysis

This chapter introduces formal verification conditions and presents an approach on how to integrate these conditions into the proposed development methodology. To justify this approach it is necessary to introduce and prove five theorems. However, when the methodology is being applied, the details of the theorems can be ignored this enables the analysts to re-use the basic theory behind the approach. The sections of this chapter concern when applying the methodology for safety specifications are 8.2 and 8.3.2, and for mission specifications are 8.4 and 8.5.2.

8.2. Safety Environment Description Analysis

The safety environment description of a system describes the properties of the sensors and actuators of the safety controller and the relationships involving these variables and the safety real world variables.

Production Guidelines

The production guidelines for an initial safety environment description of a system.

Step 1

The team must identify all *the safety controller variables* of the system, by a systematic analysis of the properties of the sensors and actuators of the safety controller. Some of the variables will represent sensors and actuators of the safety controller which are already present, others sensors and actuators which the analysts believe will be useful in the specification of the safety controller. These latter variables are determined by inspecting the safety real world specification to identify the real world variables for which a sensor or actuator may be required (and is possible to construct). An informal specification of a suitable sensor (or actuator) should then be stated. The results of this analysis are recorded as a state variables table for the sensors and actuators of the safety controller.

Step 2

The team must specify the *range* and *class* of each safety controller variable identified in *step 1*, and define the units of each variable. The results of this step are represented as a *ranges table* and a *classes table*.

Step 3

The team must identify the *invariant* and *history* relations that involve actuators or sensors of the safety controller and the real world variables. These relations fall into two broad categories.

a. Relations that express the properties of sensors (or actuators) that already exist. These relations can be determined by inspecting the specifications of the sensors (or actuators) to determine the relationship between them and the real world variables. For most sensors and actuators the relations should be straightforward. For example, a typical relation for a sensor would be an invariant relation which defines an upper bound on the difference between the value of the sensor and the real world variable being monitored. For actuators or sensors with more complex relations an iterative process of identifying a relation and then carefully checking the relation against the specification would be required.

b. Relations that express properties of sensors (or actuators) that will be developed for the safety controller. These relations can also be determined by inspecting the specifications of the sensors (or actuators) to determine the relationship between them and the real world variables. However, unlike the sensors (or actuators) that already exist the informal specifications can be modified if the relations highlight any shortcomings.

The team must also identify any initial conditions for the values of the sensors and actuators. The results of this analysis are represented as a *relations table*.

Step 4

The initial SED is defined as $SED(SY) = \langle T, Sv, VP, CP, IR, HR \rangle$, where

Sv is the sequence of real world variables and the safety controller variables;

VP the ranges of Sv ;

CP the classes of Sv ;

IR the invariant relations of SRD and the invariant relations identified in *step 3*; and

HR the history relations of SRD and the history relations identified in *step 3*.

Discussion

The introduction of the variables that represent the properties of the sensors and actuators lead to an expansion in the observable state space, that define the histories of the system. In many systems the behaviour of all the sensors (or actuators) cannot be specified without a detailed study of the safety controller requirements – a detailed study being necessary to determine what sensors and actuators are required and the properties that they should possess. In some cases, the sensors and actuators which must be used are specified as part of the requirements – for example, the customer and hazard analysts may specify that a thermometer with specific properties must be used by the safety controller. Whether all or just some of the sensors and actuators of the safety controller are specified before the controller analysis starts, to proceed to the safety controller analysis a *SED* is required. The construction guidelines (described above) lead to an initial *SED* which can be used as the basis for further analysis of the safety controller.

8.3. Safety Controller Specification Analysis

The safety controller specification of a system is expressed as an SEMG, and structured according to the general safety controller phases (see figure 8.3).

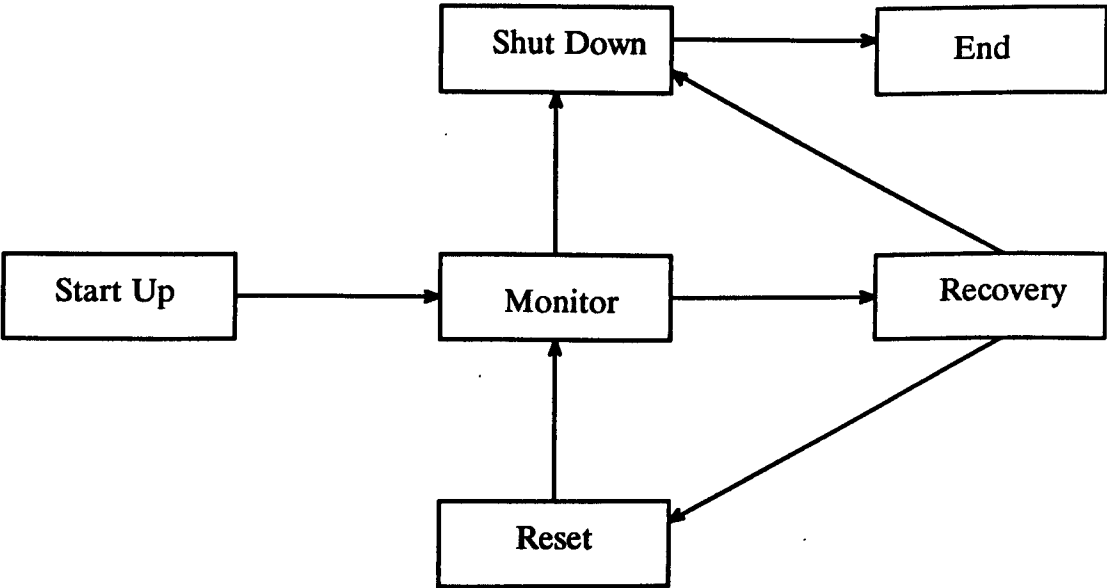


Figure 8.3. General Structure of Safety Controller

Before presenting the production guidelines, we consider the verification condition that will be used to check the produced safety controller specification. We then discuss a set of

suitable checks to confirm this verification condition. This discussion is followed by production guidelines for the safety controller specification.

8.3.1. Safety Verification.

The verification of a safety controller specification of a system against the safety real world specification of that system, involves the construction of a proof that any safety environment history that satisfies the safety controller specification must satisfy the safety real world specification.

Definition: Safety verification condition

The safety controller specification of a system is adequate if and only if the safety real world specification of that system is a consequent of the safety controller specification for the safety environment description histories of that system.

More precisely, SCS is adequate for SRS iff $\forall H \in \text{SEDH}: H \text{ sat SCS} \Rightarrow H \text{ sat SRS}$.

For a set of checks to be suitable for the confirmation of the safety verification condition (given above), they must possess the attributes described in the following paragraphs.

It must be convenient to review the proof of the safety verification condition obtained by confirming the checks. The necessity for a reviewable proof arises from the need for independent certification for safety-critical systems. Moreover, the effort required to review the verification should be considerably smaller than the effort required to perform the original verification. This would introduce the potential of several independent checks over the verification at little extra cost to that of producing the verification.

The verification checks should lead to structured proofs. Many formal verification proofs are unstructured and this makes them difficult to construct and leads to narrow thinking, but more importantly (when certification is essential) unstructured verifications are difficult to review. The verification checks should lead to a structured proof by exploiting the structure of the safety controller specification.

It must be possible to derive guidelines from the checks that enable a proof for the verification of the safety controller specification to be constructed, as the specification is being produced. The availability of such guidelines will allow the detection of *flaws* (i.e.

modes or arcs which will invalidate the satisfaction of the checks) during the production of the safety controller specification.

To allow a structured proof to be constructed, the relationship between the modes of the safety controller specification and the safety real world specification must be investigated.

Consider the satisfaction of the safety controller specification: $H \text{ sat } SCS$. From lemma 4.7. we can infer $\exists cs \in Seq(SCS): \exists t_0, \dots, t_{|cs|} \in T: H \text{ sat } cs@ \langle t_0, \dots, t_{|cs|} \rangle$.

Since SRS is a system predicate. we conclude:

$H \text{ sat } cs@ \langle t_0, \dots, t_{|cs|} \rangle \Rightarrow H \text{ sat } SRS$, if

$H \text{ sat } ms(i)@[t_{i-1}, t_i] \Rightarrow H \text{ sat } SRS@[t_{i-1}, t_i]$, for all i in $\{1, \dots, |cs|\}$.

In the next section we present a set of safety verification checks which confirm the condition above; and then prove that the safety verification checks do indeed confirm the safety verification condition. Before presenting safety verification checks, consider the following (simple) check which also confirms the condition above.

For any mode m for the mode set of SCS , any history H from $SEDH$, and any interval Int if H satisfies m during Int then H must satisfy SRS during Int .

More precisely,

$\forall m \in M(SCS): \forall H \in SEDH: \forall Int \in SI(T): [H \text{ sat } m@Int \Rightarrow H \text{ sat } SRS@Int]$.

The above check, is a simple check which allows each mode to be considered in detail, but it is too stringent. It does not exploit fact the fact that at the the start point of a mode m of SCS , a predecessor of m has completed. Hence, the above check does not take into account the state of the real world at the start of a mode. The state of the safety controller can, of course, be captured by the start and invariant predicates of a mode. It follows, that if it is possible to have a sensor for every safety real world variable then a model of the state of the real world could be captured. However in general, this will not be the case, that is there will be some real world variables that the safety controller would like to measure for which there are no sensors. In the next section, a set of verification checks are presented which allow the safety controller analysts to capture the state of the real world at the start

point of a mode m – by exploiting the fact that a predecessor of m has completed at the start point of m .

Safety Verification Checks

The safety verification checks, for a safety controller specification, are based on the construction of a *precondition function* for the safety controller specification. This precondition function defines a relationship between the modes of the safety controller specification and the behaviour of the real world. These verification checks are presented below; and are followed by a brief discussion which justifies their suitability.

Definition: Safety verification checks

We will say that a safety controller specification SCS passes the safety verification checks for a safety real world specification SRS if and only if there exists a precondition function $PF: M(SCS) \rightarrow PredSet$, and a system predicate IP such that the following three conditions hold.

i. For any history H of SEDH, H satisfies IP at the start point of the system lifetime.

More precisely, $\forall H \in SEDH: H \text{ sat } IP@s(T)$.

ii. The predicate mode graph $\langle SCS, PF \rangle$ is complete for the history description SED and system predicate IP .

More precisely, $\langle SCS, PF \rangle \text{ cmp } \langle SED, IP \rangle$.

iii. For any mode m of SCS, for any history H of SEDH and any Interval Int , if H satisfies $PF(m)$ at $s(Int)$ and m during Int then H satisfies SRS during Int .

More precisely, $\forall m \in M(SCS): \forall H \in SEDH: \forall Int \in SI(T):$

$$[H \text{ sat } PF(m)@s(Int) \wedge H \text{ sat } m@Int \Rightarrow H \text{ sat } SRS@Int].$$

The fact that SCS passes the safety verification checks for SRS will be denoted by SCS pass SRS.

To construct the first verification proof of the safety controller specification, the analysts must identify a suitable precondition function PF and system predicate IP . However, any subsequent reviews of the proof can use the PF and IP provided to test the confirmation of the three checks. Hence, the above checks are suitable for reviews.

The verification checks enable a structured proof to be constructed, by allowing the proof to be broken down into small pieces. The first check is performed over the initial predicate IP. The second check can be performed over each arc. Finally, the third check is performed over each mode.

A set of guidelines based on the verification checks, are presented in section 8.2. These guidelines lead to a proof for the confirmation of the verification checks, as the safety controller specification is being produced. These confirmations are performed using production rules derived from the three clauses of the safety verification checks.

Safety Verification Theorem

The safety verification theorem (theorem 8.1) shows that if the SCS of a system passes the safety verification checks for the SRS of that system then the safety verification condition holds for that system.

Theorem 8.1

If the safety controller specification of a system passes the safety verification checks then the safety verification condition holds for that system.

More precisely, $\text{SCS pass SRS} \Rightarrow \forall \text{ms} \in \text{SEDH}: \text{H sat SCS} \Rightarrow \text{H sat SRS}.$

Proof

This result follows from the clauses of the safety verification checks and theorem 4.4.

From clauses *i* and *ii* of the safety verification check and theorem 4.4 we have:

$$\forall \text{H} \in \text{SEDH}: \text{H sat IP@s(T)} \wedge \langle \text{SCS}, \text{PF} \rangle \text{ cmp } \langle \text{SED}, \text{IP} \rangle$$

\Rightarrow

$$\forall \text{H} \in \text{SEDH}: \forall \text{ms} \in \text{Seq}(\text{SCS}): \forall t_0, \dots, t_{|\text{ms}|} \in \text{T}:$$

$$[\text{H sat ms}@ \langle t_0, \dots, t_{|\text{ms}|} \rangle \wedge t_0 = s(\text{T}) \Rightarrow \forall i \in \{1, \dots, |\text{ms}|\}: \text{H sat PF}(\text{ms}(i))@t_{i-1}].$$

From the semantics of a mode graph we have:

$$\forall \text{H} \in \text{SEDH}: [\text{H sat SCS} \Rightarrow$$

$$\exists \text{ms} \in \text{Seq}(\text{SCS}): \exists t_0, \dots, t_{|\text{ms}|} \in \text{T}: [\text{H sat ms}@ \langle t_0, \dots, t_{|\text{ms}|} \rangle \wedge t_0 = s(\text{T}) \wedge t_{|\text{ms}|} = e(\text{T})].$$

$$\begin{aligned} \therefore \forall H \in \text{SEDH}: [& H \text{ sat } \text{SCS} \Rightarrow \\ \exists ms \in \text{Seq}(\text{SCS}): \exists t_0, \dots, t_{|ms|} \in T: \\ [H \text{ sat } ms@ \langle t_0, \dots, t_{|ms|} \rangle \wedge \forall i \in \{1, \dots, |ms|\}: & H \text{ sat } \text{PF}(ms(i))@t_{i-1}] \quad]. \end{aligned}$$

From clause *iii* of the safety verification check we have:

$$\begin{aligned} \forall m \in M(\text{SCS}): \forall H \in \text{SEDH}: \forall \text{Int} \in \text{SI}(T): \\ [H \text{ sat } \text{PF}(m)@s(\text{Int}) \wedge H \text{ sat } m@ \text{Int} \Rightarrow H \text{ sat } \text{SRS}@ \text{Int}]. \end{aligned}$$

$$\begin{aligned} \therefore \forall H \in \text{SEDH}: [& H \text{ sat } \text{SCS} \Rightarrow \\ \exists ms \in \text{Seq}(\text{SCS}): \exists t_0, \dots, t_{|ms|} \in T: \\ [t_0 = s(T) \wedge t_{|ms|} = e(T) \wedge \forall i \in \{1, \dots, |ms|\}: & H \text{ sat } \text{SRS}@[t_{i-1}, t_i] \quad]. \\ \therefore \forall H \in \text{SEDH}: [& H \text{ sat } \text{SCS} \Rightarrow H \text{ sat } \text{SRS}]. \end{aligned}$$

8.3.2 Safety Controller Specification Development Methodology

The strategy outlined in this section guides the safety controller analysts in the production of a safety controller specification, by an analysis of the safety environment description and real world specifications (i.e. SRS and MRS) of that system. There are two stages in the development strategy: *mode graph production* and *mode graph connection*.

Mode Graph Production

For each phase of the safety controller a mode graph is produced. To specify a mode graph of a phase, of the safety controller, the tasks that must be performed during that phase and the order in which they should be performed must be identified. The separation of the behaviour of the safety controller into the general phases, enhances the modularity of the safety controller specification and allows the provision of concrete guidelines. (Before the development starts clause *i* of the safety verification check must be confirmed.) Five *production* rules, are provided to check the mode graphs of the phases.

The first two rules *a* and *b* are defined below for a mode graph MG and precondition function PF; rule *a* is used to confirm clause *ii* for MG and rule *b* to confirm clause *iii* for MG. These rules relate to the arcs which connect the modes of a mode graph of a phase.

Rule a. For any arc (x, y) of $A(MG)$, any history H of $SEDH$ and any interval Int , if H satisfies the precondition of x at $s(Int)$ and x during Int and the start and invariant predicates of y at $e(Int)$ then H satisfies the precondition of y at $e(Int)$.

$\forall (x, y) \in A(MG): \forall H \in SEDH: \forall Int \in SI(T):$

$H \text{ sat } PF(x)@s(Int) \wedge H \text{ sat } x@Int \wedge H \text{ sat } (Start(y) \wedge Inv(y))@e(Int) \Rightarrow$

$H \text{ sat } PF(y)@e(Int).$

Rule b. For any mode m of MG , any history H of $SEDH$ and any interval Int , if H satisfies the precondition of m at $s(Int)$ and m during Int then H must satisfy SRS during Int .

$\forall m \in M(MG): \forall H \in SEDH: \forall Int \in SI(T):$

$H \text{ sat } PF(m)@s(Int) \wedge H \text{ sat } m@Int \Rightarrow H \text{ sat } SRS@Int.$

The rule c is defined below for a pair of modes (x, y) and precondition function PF ; rule c is used to confirm clause ii for (x, y) . This rule and rules d and e are related to the arcs that connect the mode graphs of two phases.

Rule c. For any history H of $SEDH$ and any interval Int , if H satisfies the precondition of x at $s(Int)$ and x during Int and the start and invariant predicates of y at $e(Int)$ then H satisfies the precondition of y at $e(Int)$.

$\forall H \in SEDH: \forall Int \in SI(T):$

$H \text{ sat } PF(x)@s(Int) \wedge H \text{ sat } x@Int \wedge H \text{ sat } (Start(y) \wedge Inv(y))@e(Int) \Rightarrow H \text{ sat } PF(y)@e(Int).$

The rules d and e , are defined below for a pair of modes (x, y) rule d is used to confirm the completeness of (x, y) and rule e to confirm the consistency of (x, y) .

Rule d. For any history H of $SEDH$ and any interval Int , if H satisfies x during Int then H satisfies the the start and invariant predicates of y at $e(Int)$.

$\forall H \in SEDH: \forall Int \in SI(T): [H \text{ sat } x@Int \Rightarrow H \text{ sat } (Start(y) \wedge Inv(y))@e(Int)].$

Rule e. There exists a history H of $Iset(SED)$ and a time point t such that H satisfies the conjunction of the invariant and end predicates of x , and the start, invariant and negation of the end predicate of y .

$\exists H \in Iset(SED): \exists t \in T: H \text{ sat } (Inv(x) \wedge End(x) \wedge Start(y) \wedge Inv(y) \wedge \neg End(y))@t.$

Before the analysis of the phases can start a system predicate that satisfies clause *i* of the safety verification checks must be identified; and designated as the initial predicate (IP). A suitable system predicate can be derived directly from the conditions imposed on the start point of the system lifetime by the SEDH of the system. For, each phase the last one or two steps require the construction of proofs for the production rules. If proofs cannot be constructed then the specification produced, by following the preceding steps of the phase must be modified. Guidelines on modifications are given in step 6.

Step 1 (Start up phase)

The behaviour of the safety controller during the start up phase is specified by a start up graph, denoted by SU. There are six steps in the start up phase.

- a.* Define a basic strategy for the start up phase; and identify the tasks that must be performed by the safety controller, to realise the strategy.
- b.* Consider, how the tasks identified above can be specified by modes; and sketch a mode graph for the start up phase. This sketch must show the interactions between the modes; and provide informal specifications for the modes.
- c.* Extend, the precondition function (PF) for the mode graph sketched in *step 1.b*. The precondition of the mode graph can be determined in two stages.
 - i.* For each mode m of the mode graph, analyse the informal specification of m to identify the conditions that must hold over the real world at the start of m for rule b to hold for a formal specification constructed for m . These conditions should then be defined as a system predicate (and denoted by $PC(m)$) and the assertion $PF(m) \Rightarrow PC(m)$ stated.
 - ii.* Check if a precondition function can be constructed which will satisfy the assertions stated above and pass rule *a*. The precondition of a mode m can be checked by investigating the behaviour of the state variables (over which the precondition is defined) during the predecessors of m . This investigation is performed by an analysis which checks if for each predecessor p the condition which holds over the real world at the end of p and the conditions over the safety controller at the start of m are sufficient to ensure $PC(m)$ will hold at the start of m . (The case of the mode $S(SU)$ is special since it has no predecessor, for

this mode it must be confirmed that $IP \Rightarrow PC(S(SU))$.) If this analysis determines that a suitable system predicate SP can be defined for m , SP is suitable if SP will pass rule a for all arcs leading to m and $SP \Rightarrow PC(m)$, define $PF(m)$ as SP. However, if the analysis determines that a suitable system predicate cannot be defined then the specification of either m or a predecessor of m must be modified.

d. Construct the mode graph SU. The mode specification of a mode is constructed by an analysis of its informal specification.

e. Construct proofs for the consistency and completeness of SU.

f. Construct proofs for the rules a and b for SU and PF; and confirm $IP \Rightarrow PF(S(SU))$.

Step 2 (Monitor Phase)

The behaviour of the safety controller during the monitor phase is specified by a monitor graph, denoted by MN, with a single start mode. The monitor graph is connected to the start up phase by connecting the end mode of the start up graph to the start mode of the monitor graph.

The mode graph for the monitor phase is developed in seven steps.

a. Construct a phase graph, defined over the safety real world variables. This phase graph must specify the behaviour exhibited by the physical process as measured by the safety real world variables, during a typical history that satisfies the MRS.

b. Modify the phase graph, defined over the sensors of the safety controller. This phase graph PS must reflect the perception, by the safety controller, of the behaviour exhibited by the physical process as it passes through the phases defined in *step 2.a*.

c. For each phase, of PS, construct mode graphs (that have a single start mode) which define the behaviour of the safety controller. The mode graph constructed for a phase must ensure that the SRS is maintained by the safety controller, regardless of the behaviour of the mission controller, but also allow the mission controller to complete the task being monitored by the safety controller during the phase. The construction of the mode graph and precondition of the mode graph, for a phase consist of four stages.

- i.* Define a basic strategy for the phase; and identify the tasks that must be performed by the safety controller, to realise the strategy.
 - ii.* Consider how the tasks identified above can be specified by modes; and sketch a mode graph for the phase.
 - iii.* Extend the precondition function (PF) for the mode graph sketched in the previous step. The precondition of the mode graph can be determined in two stages (see step 1.c).
 - iv.* Construct the mode graph of the phase. The mode specification of a mode is constructed by an analysis of its informal specification.
- d.* The mode graphs constructed in *step 2.c* are then combined to construct the monitor graph MN. Roughly speaking, MN is defined by performing the next four steps.
- i.* The mode set is the union of the mode sets of the mode graphs for the phases of PS.
 - ii.* The arc set is the union of the arc sets of the mode graphs for the phases of PS and for every pair of phases (x, y) in the arc set, of PS, add an arc from each end mode of the mode graph for x to the start mode of the mode graph for y .
 - iii.* The start mode is the start mode of the mode graph for $S(PS)$.
 - iv.* The end mode set is the end mode set of the mode graph for $E(PS)$.
- In constructing the monitor graph conservative decisions must be made to ensure safety, however care must be taken that the decisions are not so stringent that it becomes impossible for the mission controller to achieve the mission.
- e.* Construct proofs for the consistency of MN.
 - f.* Construct proofs for the rules *a* and *b* for MN and function PF.
 - g.* Construct proofs for the rules *c*, *d* and *e* for the pair $(E(SU), S(MN))$ and function PF.

Step 3 (Recovery Phase)

The behaviour of the safety controller during the recovery phases is specified by SEMGs, denoted by a function REC, the recovery graph of mode m is denoted by $REC(m)$. $REC(m)$ is connected to the monitor phase by constructing an arc from m to the start mode of $REC(m)$. A seven step guide to the construction of the recovery graph is given below.

a. Identify all the monitor modes for which a recovery graph is necessary. A recovery graph is not needed for a mode if after the end of the mode the safety controller will remain in the monitor phase. Hence, a mode m of the monitor mode does not need a recovery graph if it satisfies the following condition:

$$\forall H \in \text{SEDH}: \forall \text{Int} \in \text{SI}(T): H \text{ sat } m @ \text{Int} \rightarrow \exists x \in \text{MN.sr}(m): H \text{ sat } \text{Start}(x) \wedge \text{Inv}(x) @ e(\text{Int}).$$

(A mode that satisfies the above condition is complete, see section 4.3.3).

A recovery graph must be constructed for the set of modes that do not satisfy the above condition, this set will be denoted by REM.

b. Specify the start predicate of the start mode of the recovery graph of each mode in the set REM. The safety controller must enter the recovery phase at the end of a mode m only when the start predicate of a successor of m is not satisfied at the end of m . Hence we define $\text{Start}(\text{REC}(m))$ as the conjunction of the end predicate to the negation of the disjunction of the start predicates of the successors of m .

$$\text{Start}(\text{REC}(m)) := \text{End}(m) \wedge \neg (\bigvee x \in \text{MN.sr}(m): \text{Start}(x) \wedge \text{Inv}(x)).$$

c. Define a basic strategy for each recovery graph; and identify the tasks that must be performed by the safety controller, to realise the strategy.

d. The following three steps are repeated for each recovery graph (i.e. m in REM).

i. Consider how the tasks of $\text{REC}(m)$ can be specified by modes; and sketch a mode graph.

ii. Extend PF for the mode graph sketched in the previous step (see *step 1.c*).

iii. Construct the mode graph $\text{REC}(m)$.

e. Construct proofs for the consistency and completeness of the recovery graphs.

f. Construct proofs for the rules *a* and *b* for $\text{REC}(m)$ and the function PF, for all m in REM.

g. Construct proofs for the rules *c*, *d* and *e* for the pair $(m, \text{S}(\text{REC}(m)))$ and the function PF, for all m in REM.

Step 4 (Reset Phase)

The behaviour of the safety controller during the reset phases is specified by reset graphs, the reset graphs are denoted by a function RG, the reset graph of mode m is

denoted by $RG(m)$. The mode to which the end of $RG(m)$ is connected will be denoted by $Con(m)$. Hence, $RG(m)$ is connected into SCS by the construction of an arc from m to $S(RG(m))$, and an arc from the $E(RG(m))$ to $Con(m)$.

The reset graphs are constructed in six steps.

- a.* Identify the recovery graphs for which a reset graph will be defined. A reset graph is constructed for the end mode of a recovery graph only if the system is allowed to enter the monitor phase after the recovery represented by the associated recovery graph has taken place. The set of modes for which a reset graph will be constructed are denoted by RGM .
- b.* Define a basic strategy for each reset graph, these strategies must also define the mode that follows the reset graph. For each strategy identify the tasks that must be performed by the safety controller.
- c.* The following three steps are repeated for each reset graph (i.e. m in RGM).
 - i.* Consider how the tasks of $RG(m)$ can be specified by modes; and sketch a mode graph.
 - ii.* Extend PF for the mode graph of the previous step (see *step 1.c*).
 - iii.* Construct the mode graph $RG(m)$.
- d.* Construct proofs for the consistency and completeness conditions for the reset graphs.
- e.* Construct proofs for the rules *a* and *b* for $RG(m)$ and the function PF, for all m in RGM .
- f.* Construct proofs for the rules *c*, *d* and *e* for the pairs $(E(REC(m)), RG(m))$ and $(E(RG(m)), Con(m))$ with the function PF, for all m in RGM .

Step 5 (Shut Down Phase and End Phase)

Recall that the behaviour of the safety controller during the shut down phase is specified by shut down graphs; these are denoted by the function SH , the shut down graph for mode m is denoted by $SH(m)$; and the end controller mode is denoted by EC . A shut down graph for a mode m is connected into the safety controller specification by the construction of an arc from m to $SH(m)$ and an arc from $E(SH(m))$ to EC .

- a.* This step consists of four stages.
 - i.* Construct the end controller mode EC . The end controller mode must specify a task

during which the safety controller holds that actuators constant, and the SRS is maintained. Such a mode should be specified.

ii. Define $PF(EC)$. By analysing EC, to identify the behaviour necessary at the start point of EC for SRS to be maintained during EC, $PF(EC)$ must be defined.

iii. Define the shutdown condition. The shut down condition will usually be a state of a selector on the safety operator console.

iv. Construct a proof for rule b for EC.

b . Identify the modes from which a shutdown graph starts. A shutdown graph is required for a monitor mode (or the end mode of a recovery graph) if and only if the safety operator is to have the option to start the shutdown of the safety controller while the safety controller is performing the task described by that mode. The set of modes for which a shutdown graph is constructed are denoted by SHM. Then the end predicates of the shut down modes must be modified by the disjunction of the shut down condition.

c . The following three steps are repeated for each shut down graph.

i. Consider, how the tasks of $SH(m)$ can be specified by modes; and sketch a mode graph.

ii. Extend, PF for the mode graph of the previous step (see *step 1.c*).

iii. Construct the mode graph $SH(m)$.

d . Construct proofs for the consistency and completeness of the shut down graphs.

e . Construct proofs for the rules a and b for $SH(m)$ and the function PF , for all m in SHM.

f . Construct proofs for the rules c , d and e for the pairs $(m, S(SH(m)))$ and $(E(SH(m)), EC)$ with the function PF , for all m in SHM.

Step 6 (Modifications)

This step must be performed if a mode m does not pass a *production* rule during the production of the mode graph for the safety controller phase P . Four options are outlined below, in the order that they should be attempted.

a . *Modify mode specification*. This should be attempted first since it requires the minimal changes, to the SCS. The mode specification of m must be checked in the light of the

production rule that is violated, to identify how the mode specification can be modified to allow m to pass that rule. Once the mode specification of m has been modified any production rules which were checked over m , must be confirmed.

b. Modify PF. The PF definition over the modes of P must be checked in the light of the production rule that is violated, to identify how PF can be modified to allow m to pass that rule. The main drawback of modifying PF is that those rules involving the modes for which the precondition is modified must be confirmed again.

c. Modify mode graph. The mode graph of P must be checked in the light of the production rule that is violated, to identify how the mode graph can be modified to allow it to pass the production rules of P . The main drawback of modifying the mode graph of P is that PF must be defined over the new mode graph; and the production rules for P checked.

d. Modify SED. Unlike, the first three options, the safety controller analysts do not have direct control over the relationships of the SED. The safety controller analyst should identify the relations that would be required for m to pass the production rule. If the analysts are fortunate these additional relation may hold for the sensors or actuators, otherwise modifications to the sensors or actuators or changes to the physical process would be required. Modifications to the SED, can lead to the need to check rules of phases other than P .

If for a particular case, a simple fix can be seen by any of the options given above then it should of course be performed directly.

Mode Graph Connection

The mode graphs of the phases are connected together to construct the safety controller specification, by the transformation CSCS.

Algorithm 8.1

Function CSCS(SU, MN: SEMGraph; REC, RG, SH: GraphFunction; Con: ModeFunction; REM, RM, SHM: SetOfModes; EC: Mode): SEMGraph;

Var MG: SEMGraph;

m: Mode;

MREC, MRG, MSH: SetOfModes;

AMN, AREC, ARG, ASH: SetOfArcs;

1: MREC := $\bigcup_{m \in \text{REM}} M(\text{REC}(m))$;

2: MRG := $\bigcup_{m \in \text{RGM}} M(\text{RG}(m))$;

3: MSH := $\bigcup_{m \in \text{SHM}} M(\text{SH}(m))$;

4: $M(\text{MG}) := M(\text{SU}) \cup M(\text{MN}) \cup \text{MREC} \cup \text{MRG} \cup \text{MSH} \cup \{\text{EC}\}$;

5: $\text{AMN} := A(\text{MN}) \cup \{(E(\text{SU}), S(\text{MN}))\}$;

6: $\text{AREC} := \bigcup_{m \in \text{REM}} [A(\text{REC}(m)) \cup \{(m, S(\text{REC}(m)))\}]$;

7: $\text{ARG} := \bigcup_{m \in \text{RGM}} [A(\text{RG}(m)) \cup \{(m, S(\text{RG}(m)))\}] \cup \{(E(\text{RG}(m)), \text{Con}(m))\}$;

8: $\text{ASH} := \bigcup_{m \in \text{SHM}} [A(\text{SH}(m)) \cup \{(m, S(\text{SH}(m)))\}] \cup \{(E(\text{SH}(m)), \text{EC})\}$;

9: $A(\text{MG}) := A(\text{SU}) \cup \text{AMN} \cup \text{AREC} \cup \text{ARG} \cup \text{ASH}$;

10: $S(\text{MG}) := S(\text{SU})$;

11: $E(\text{MG}) := \text{EC}$;

12: $\text{CSCS} := \text{MG}$;

13: **Stop.**

Comments

1: The set of modes of the recovery phase (MREC) is the union of the mode sets of the recovery graphs.

2: The set of modes of the reset phase (MRG) is the union of the mode sets of the reset graphs.

3: The set of modes of the shut down phase (MSH) is the union of the mode sets of the shut down graphs.

4: The *mode set* of MG is the union of the mode set of the start graph and monitor graph, and the set of modes of the recovery phase, reset phase, shut down phase and end controller mode.

5: The set of arcs of the monitor phase (AMN) is the union of the arc set of the monitor mode and an arc that connects the end mode of the start up graph to the start mode of the monitor graph.

6: The set of arcs of the recovery phase (AREC) is the union of the arc sets of the recovery

graphs and for all m in the set REM the arcs which connect m to $S(\text{REC}(m))$.

7: The set of arcs of the reset phase (ARG) is the union of the arc sets of the reset graphs and for all modes m in RGM the arcs which connect m to $S(\text{RG}(m))$ and $E(\text{RG}(m))$ to $\text{Con}(m)$.

8: The set of arcs of the shutdown phase (ASH) is the union of the arc sets of shut down graphs and for all modes m in SHM the arcs which connect m to $S(\text{SH}(m))$ and $E(\text{SH}(m))$ to EC.

9: The *arc set* of MG is the union of the arc set of the start up graph, the set of arcs of the monitor phase, recovery phase, reset phase and shut down phase.

10: The *start mode* of MG is the start mode of the start up graph.

11: The *end mode* of MG is the end controller mode.

Safety Production Checks.

The safety controller analysts must confirm that *SED* is an accurate representation of the relationship over the real world variables and between the safety controllers sensors, actuators and real world. This confirmation should be performed by a careful systematic check of all the description relations. In particular, any relations which are introduced during the confirmation of production rules should be checked.

If the guidelines for the construction of the SCS, have been followed rigoursly and all proofs constructed for the production rules, then from theorem 8.2 (see section 8.3.3) it follows that SCS will pass the safety verification checks. However, it is possible that the guidelines have not been followed completely (i.e., some proofs have been omitted and some steps missed). Even if all the steps of the guidelines have been performed, there may be errors in the proofs. Because of these difficulties, the SCS must be independently verified against the SRS. This independent verification can use the PF and IP produced during the production of SCS. In addition, the safety controller analysts must indicate the relations that are used to confirm the productions rule for a mode or arc. The independent verification should then attempt to confirm the safety verification checks, using this PF and

IP and for each mode or arc only the relations indicated by the controller analysts should be used.

Similarly, the completeness and consistency of SCS should be confirmed independently.

8.3.3. Safety Controller Specification Methodology Theorems

In this section I will present two theorems that show the production guidelines are sufficient to produce a safety controller specification that will satisfy the safety verification checks, and is both complete and consistent.

Safety Production Theorem

The safety production theorem (theorem 8.2) shows that the mode graph (SCS) produced by applying the function CSCS to the mode graphs, Graph function and sets produced by following the production guidelines will pass the safety verification check. Hence, (from theorem 8.1) such a SCS will satisfy the safety verification condition.

Theorem 8.2

If clause i of the safety verification checks is passed for a system predicate IP and the mode graphs SU and MN, the functions REG (defined over REM), RG, Con (defined over RM) and SH (defined over SHM); and mode EC pass the production rules a, b and c (as given in the production guidelines) for the safety real world specification SRS, the precondition PF and the initial predicate IP then the mode graph $SCS = CSCS(SU, MN, REG, RG, SH, Con, REM, RM, SHM, EC)$, passes the safety verification checks for SRS for PF and IP.

Proof.

The proof is given by showing how the three clauses of the safety verification checks follow from the production guidelines.

Clause i. $\forall H \in SEDH: H \text{ sat } IP@s(T)$, follows from confirmation of IP by initial check.

Clause ii. $\langle SCS, PF \rangle \text{ cmp } \langle SED, IP \rangle$.

The definition of a complete predicate mode graph (see section 4.3.7) is recalled below.

$\langle MG, PF \rangle \text{ cmp } \langle D, SP \rangle$ iff

i) $\forall (x, y) \in A(MG): \forall H \in \text{Set}(D): \forall \text{Int} \in SI(T):$

$[H \text{ sat } PF(x)@s(Int) \wedge H \text{ sat } x@Int \wedge H \text{ sat } (Start(y) \wedge Inv(y)) @e(Int) \Rightarrow H \text{ sat } PF(y)@e(Int)]$; and

ii) $\forall H \in Set(D): \forall t \in T: [H \text{ sat } SP@t \Rightarrow H \text{ sat } PF(S(MG))@t]$.

Hence to prove $\langle SCS, PF \rangle \text{ cmp } \langle SED, IP \rangle$ we must show the proof the following two conditions.

i. $\forall (x, y) \in A(SCS): \forall H \in SEDH: \forall Int \in SI(T):$

$[H \text{ sat } PF(x)@s(Int) \wedge H \text{ sat } x@Int \wedge H \text{ sat } (Start(y) \wedge Inv(y)) @e(Int) \Rightarrow H \text{ sat } PF(y)@e(Int)]$.

$A(SCS)$, is defined by step 9 of CSCS: $A(SCS) := A(SU) \cup AMN \cup AREC \cup ARG \cup ASH$.

The fact that the pairs $(x, y) \in A(SU)$, satisfy condition *i*, follows directly from the fact that the arcs of SU pass rule *a* (see step 1.f of production guidelines).

AMN is defined by step 5 of algorithm 8.1: $AMN := A(MN) \cup \{(E(SU), S(MN))\}$.

The fact that the pairs $(x, y) \in AMN$, satisfy condition *i*, follows directly from the fact that the arcs of MN pass rule *a* and the arc $(E(SU), S(MN))$ passes rule *c* (see step 2.g of production guidelines).

Arguments similar to those given for the arcs in AMN can be given for the arcs in the sets $AREC$, ARG and ASH .

ii. $\forall H \in SEDH: \forall t \in T: [H \text{ sat } IP@t \Rightarrow H \text{ sat } PF(S(SCS))@t]$.

$S(SCS)$, is defined by step 10 of algorithm 8.2: $S(SCS) := S(SU)$.

The fact that $S(SU)$ is consequent of IP follows directly from step 1.f, of production guidelines.

Clause *iii*. $\forall m \in M(SCS): \forall H \in SEDH: \forall Int \in SI(T):$

$[H \text{ sat } PF(m)@s(Int) \wedge H \text{ sat } m@Int \Rightarrow H \text{ sat } SRS@Int]$.

$M(SCS)$ is defined by step 4 of algorithm 8.2:

$M(SCS) := M(SU) \cup M(MN) \cup MREC \cup MRG \cup MSH \cup \{EC\}$

The fact that clause *iii* holds for the modes in the set $M(SU)$, follows from the fact that the modes of SU pass rule *b* (see step 1.f of production guidelines). A similar argument can be given for the modes in the set $M(MN)$, $MREC$, MRG and MSH , and the mode EC .

Mode Graph Production Theorem

The mode graph production theorem (theorem 8.3) shows that the mode graph (SCS) produced by applying the function CSCS to the mode graphs, graph functions and sets produced by following the production guidelines will be complete and consistent.

Theorem 8.3

If the mode graphs SU and MN are complete and consistent and the mode graph given by functions REG (defined over REM), RG (defined over RM) and SH (defined over SHM) are complete and consistent, mode EC is consistent; and pass production rules d and e (as given in the guidelines) then the mode graph $SCS = CSCS(SU, MN, REG, RG, SH, Con, REM, RM, SHM, EC)$, is complete and consistent.

Proof.

The proof is given by showing how the completeness and consistency follow from the production guidelines.

Completeness

We must show that all the modes of SCS are complete modes. $M(SCS)$ is defined by step 4 of algorithm 8.1: $M(SCS) := M(SU) \cup M(MN) \cup MREC \cup MRG \cup MSH \cup \{EC\}$. Consider the set of modes $M(SU)$. The modes in the set $M(SU) - \{E(SU)\}$ are complete since SU is complete (see step 1.e of the production guidelines), and mode $E(SU)$ is complete since rule d holds for the arc $(E(SU), S(MN))$ (see step 2.g of production guidelines). Consider the set of modes $M(MN)$, these modes are either complete or in the set REM (see step 3.b of production guidelines). Those modes in the set REM are made complete by constructing suitable recovery graphs (see step 3.g). Similar arguments can be given for the sets, $MREC$, MRG and MSH . The mode EC is complete since it has no successor.

Consistency

We must show that all the arcs of SCS are consistent arcs. $A(SCS)$ is defined by step 9 of algorithm 8.2: $A(SCS) := A(SU) \cup AMN \cup AREC \cup ARG \cup ASH$. Consider the set of arcs $A(SU)$, these are consistent since SU is consistent (see step 1.e of the production

guidelines). Consider the set of arcs AMN, AMN is defined by step 5 of algorithm 8.1:

$$AMN := A(MN) \cup \{(E(SU), S(REC(m)))\}.$$

The fact that the pairs $(x, y) \in AMN$ are consistent, follows directly from the fact that MN is consistent and the arc $(E(SU), S(MN))$ passes rule *e* (see step 2.g of production guidelines). Arguments similar to those given for the arcs in AMN can be given for the arcs in the sets AREC, ARG and ASH.

8.4. Mission Environment Description Analysis

The *MED* of a system is built as an extension to the *MRD* of that system. The description is extended by the introduction of the variables which represent the properties of the sensors and actuators of the mission controller. The description must also include the variables of the safety controller. The introduction of the mission controller variables expands the observable state space at the controller level. To avoid relationships in which the mission controller influences the safety controller, only two classes of relations are allowed: the *mission* relations and *dependent* relations (see chapter 6).

Construction Guidelines

The guidelines for the production of the mission environment description of a system are presented in two stages: *basic description* and *monitor relations*.

Basic Description

Step 1

The team must identify all the *mission controller variables* of the system, by an analysis of the mission controller. The analysis process is similar to that performed for the safety controller.

Step 2

The team must identify the range and class of each variable identified in step 1, and define the units of each variable.

Step 3

The team must identify the *invariant* and *history* relations that involve actuators or

sensors of the mission controller and the real world variables. These relations are determined in the same way as the invariant and history relations of the SED. The analysis in this step should consider the mission controller in isolation (i.e. it is not necessary to take into account the influence of the safety controller in this step).

Step 4

The team must determine if any of the invariant or history relations that involve actuators or sensors of the mission controller are influenced by safety controller variables. This can be determined by systematically checking if an actuator of the safety controller can influence the relationships. For each mission relation, which can be influenced by a safety controller variable a dependent relation is constructed which replaces the mission relation.

Step 5

The initial MED is defined as $MED(SY) = \langle T, Sv, VP, CP, IR, HR \rangle$, where

Sv is the sequence of real world variables and controller variables;

VP the ranges of Sv ;

CP the classes of Sv ;

IR the invariant relations of MRD and the invariant relations identified in *step 3 and 4*; and

HR the history relations of MRD and the history relations identified in *step 3 and 4*.

Monitor Relations

Step 1 (Monitor Phase Relations)

a. Make a list (denoted by DV) of all the safety controller variables that influence the mission controller. These variables can be identified by looking up the related variables column in MED.

b. Identify any invariant or history relations which hold over the variables in the list DV . There are two main types of relations. Firstly, the relations that follow from the invariant of the monitor graph, these will be invariant relations. The identification of these invariant relations, involves a simple inspection of the invariant of the monitor graph. Secondly, the relations that follow from the invariant of modes in the monitor graph, these will be history

relations. These relations can be identified by inspecting the invariant and end predicates of the modes.

Step 2 (Monitor controller variables)

a. Determine if it is necessary for the mission controller to monitor the safety controller. This can be checked by inspecting the relations of MED; and the relations identified in step 1.

b. If the analysts determine that the mission controller must monitor the safety controller. Then suitable sensor variables and relations must be defined.

Step 3 (Modify Description)

The MED produced by *step 5* of the basic description guidelines is modified by adding the variables identified in *step 1.a* and *step 2.b* to the variable sequence of MED; and the relations identified in *step 1.b* and *step 2.b* to the relations of MED.

8.5. Mission Controller Specification Analysis

The mission controller of a system is expressed as an SEMG, the structure of this graph is based on the structure of the mission real world specification of that system. As was pointed out in section 6.4.1, a natural approach to the development of the mission controller specification, is based on the construction of SEMGs for the modes of the mission real world specification. The relationship between the modes of the mission real world specification and the SEMGs is captured by the definition of a *controller function* for the mission real world specification.

Definition: Controller function

A controller function CF of a mission real world specification is a function from the modes of the mission real world specification to a set of SEMGs, such that the SEMG constructed for mode m is given by $CF(m)$ and the mode sets of the graphs given by the function are disjoint.

A function that connects the SEMGs which are specified by a controller function is defined below.

Algorithm 8.2

Function MCGT(MG: SEMGraph; CF: GraphFunction): SEMGraph;

Var MC: SEMGraph;

m, w, x, y, z: Modes;

IA, EA: SetOfModes;

1: $M(MC) := \bigcup_{m \in M(MG)} M(CF(m))$;

2: $IA := \bigcup_{m \in M(MG)} A(CF(m))$;

3: $EA := \{(w, x) \in M(MC)^2 \mid \exists (y, z) \in A(MG) \wedge w = E(CF(y)) \wedge x = S(CF(z))\}$;

4: $A(MC) := IA \cup EA$;

5: $S(MC) := S(CF(S(MG)))$;

6: $E(MC) := E(CF(S(MG)))$;

7: $MCGT := MC$;

8: **Stop.**

Comments

- 1: The *mode set* of MC is the union of the mode sets of the mode graphs given by CF.
- 2: The *internal arc set* of MC (i.e. the arcs of the controller graph) is the union of the arc sets of the mode graphs given by CF.
- 3: The *external arc set* of MC (i.e., the arcs that are used to connect the controller graphs) is the set of all pair of modes (w, x) from set $M(MC)^2$ for which there is an arc (y, z) in MG such that the end mode of the controller graph of y is w and the start mode of the controller graph of z is x.
- 4: The *arc set* of MC is the union of the internal and external arc sets.
- 5: The *start mode* of MC is the start mode of the controller graph of the start mode of MG.
- 6: The *end mode* of MC is the end mode of the controller graph of the end mode of MG.

Lemma 8.1

For any mode sequence cs from the set $Seq(MCGT(MRS, CF))$ there is a mode sequence rs of $Seq(MRS)$ and an increasing function $k: \{0, \dots, |rs|\} \rightarrow \{0, \dots, |cs|\}$ such that $k(0) = 0$ and $k(|rs|) = |cs|$ and $cs(k(i-1) + 1, k(i))$ is an element of $Seq(CF(rs(i)))$, for all i in $\{1, \dots, |rs|\}$.

More precisely,

$\forall cs \in \text{Seq}(\text{MCGT}(\text{MRS}, \text{CF}))$:

$\exists rs \in \text{Seq}(\text{MRS})$: $\exists k \in \text{KF}$: $\forall i \in \{1, \dots, |rs|\}$: $cs(k(i-1)+1, k(i)) \in \text{Seq}(\text{CF}(rs(i)))$,

where KF denotes the set of functions $k: \{0, \dots, |rs|\} \rightarrow \{0, \dots, |cs|\}$ such that k is increasing, $k(0)=0$ and $k(|rs|)=|cs|$.

Proof.

Consider any sequence cs of $\text{Seq}(\text{MCGT}(\text{MRS}, \text{CF}))$ (of finite length), in the following we show that there exists a sequence rs of $\text{Seq}(\text{MRS})$ and a sequence $k_0, \dots, k_{|rs|}$ that satisfies this lemma. From step 5, of algorithm 8.1 we have $cs(1) = S(\text{CF}(S(\text{MRS})))$, in the rest of this proof we will denote $S(\text{MRS})$ by r_1 and let $k_0=0$. From steps 2, 3 and 4, of algorithm 8.1, $\exists k_1$: $cs(k_0, k_1) \in \text{Seq}(\text{CF}(S(r_1)))$ and from step 6, of algorithm 8.2, if $k_1 = |cs|$, $r_1 = E(\text{MRS})$, otherwise $\exists r_2 \in M(\text{MRS})$: $(r_1, r_2) \in A(\text{MRS}) \wedge cs(k_1+1) = S(\text{CF}(S(r_2)))$. The previous argument can be repeated for $r_i, i = \{2, \dots, q\}$, where $k_q = |cs|$; such a q must exist since $k_i > k_{i-1}$ and cs is of finite length. Hence:

$\exists r_1, \dots, r_q \in M(\text{MRS})$: $\forall i \in \{1, \dots, q-1\}$: $(r_i, r_{i+1}) \in A(\text{MRS}) \wedge r_1 = S(\text{MRS}) \wedge r_q = E(\text{MRS})$.

$\therefore \langle r_1, \dots, r_q \rangle \in \text{Seq}(\text{MRS})$. To complete the proof we denote $\langle r_1, \dots, r_q \rangle$ by rs (i.e., $q = |rs|$), and define $k(i) = k_i$, for all $i \in \{0, \dots, |rs|\}$. Hence k is an increasing function, $k(0)=0$ and $k(|rs|) = |cs|$, that is $k \in \text{KF}$.

$\therefore \exists rs \in \text{Seq}(\text{MRS})$: $\exists k \in \text{KF}$: $\forall i \in \{1, \dots, |rs|\}$: $cs(k(i-1)+1, k(i)) \in \text{Seq}(\text{CF}(rs(i)))$.

The relationship between k , cs and rs is illustrated in figure 8.4.

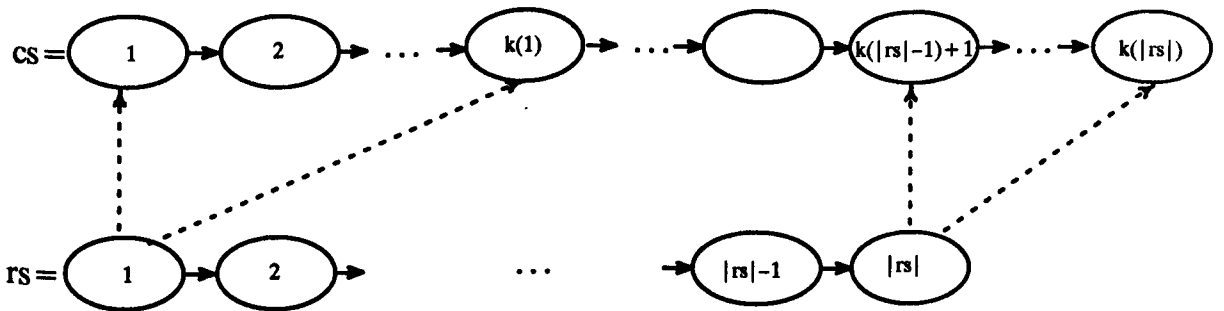


Figure 8.4. Picture of Lemma 8.1.

Before presenting the production guidelines, we consider the verification condition that will be used to check the produced mission controller specification. We then discuss a

set of suitable checks to confirm this verification condition. This discussion is followed by production guidelines for the mission controller specification, which are presented as the stages shown in figure 8.2.

8.5.1 Mission Verification.

The verification of the mission controller specification of a system, involves the construction of a proof that any mission environment history that satisfies the mission controller specification must satisfy the mission real world specification of that system

Definition: Mission verification condition

The mission controller specification of a system is adequate if and only if the mission real world specification of that system is a consequent of the mission controller specification for the mission environment histories of that system.

More precisely, MCS is adequate for MRS iff $\forall H \in \text{MEDH}: H \text{ sat MCS} \Rightarrow H \text{ sat MRS}$.

For a set of checks to be suitable for the verification condition they must posses the same attributes that were suggested for the safety verification condition. In addition, any verification checks should not be too stringent, since one motivation for separation of the safety from the mission is to allow more freedom in the specification of the mission controller.

To allow a structured proof to be constructed, the verification checks must exploit the relationship between the mission controller specification and mission real world specifications as captured by lemma 8.1.

Consider the satisfaction of the mission controller specification: $H \text{ sat MCS}$. From lemma 4.7, we can infer:

$$\exists cs \in \text{Seq}(\text{MCS}): \exists t_0, \dots, t_{|cs|} \in T: H \text{ sat } cs@ \langle t_0, \dots, t_{|cs|} \rangle \wedge t_0 = s(T) \wedge t_{|cs|} = e(T).$$

From lemma 8.1 we can infer $\exists rs \in \text{Seq}(\text{MRS}): \exists k \in \text{KF}: cs(k(i-1) + 1, k(i)) \in \text{Seq}(\text{CF}(rs(i)))$.

Hence, $\exists k \in \text{KF}: \forall i \in \{1, \dots, |rs|\}: H \text{ sat } \text{CF}(rs(i))@[t_{k(i-1)}, t_{k(i)}]$.

Postulated Checks

Next we postulate checks which will imply $H \text{ sat } cs \Rightarrow H \text{ sat } rs$. The general approach is based on demonstrating that the controller graph of a mode $rs(i)$, ensures $rs(i)$ ends before

the end point of the controller graph, and that $rs(i+1)$ starts after the end point of $rs(i)$. The modes $rs(1)$ and $rs(|rs|)$ are treated as special cases. The controller graph of $rs(1)$ must ensure $rs(1)$ is satisfied during the satisfaction of the controller graph and $rs(2)$ starts after the end point of $rs(1)$, and the controller graph of $rs(|rs|)$ must ensure that $rs(|rs|)$ ends at the end point of the controller graph. The checks are given below, and illustrated in figure 8.5.

From lemmas 4.7 and 8.1 we have:

$$\exists cs \in Seq(MCS): \exists t_0, \dots, t_{|cs|} \in T: H \text{ sat } cs@ \langle t_0, \dots, t_{|cs|} \rangle \wedge t_0 = s(T) \wedge t_{|cs|} = e(T) \text{ and}$$

$$\exists k \in KF: \forall i \in \{1, \dots, |rs|\}: H \text{ sat } CF(rs(i))@[t_{k(i-1)}, t_{k(i)}].$$

Next we give three conditions over the time points $t_0, \dots, t_{|cs|}$ and function K .

a. There exists a time point e_1 in the interval $[t_{k(0)}, t_{k(1)}]$ such that H satisfies $rs(1)$ during $[t_{k(0)}, e_1]$ and H starts $rs(2)$ during $[e_1, t_{k(1)}]$.

More precisely,

$$\exists e_1 \in [t_{k(0)}, t_{k(1)}]: H \text{ sat } rs(1)@[t_{k(0)}, e_1] \wedge H \text{ sat } \mathcal{J}(rs(2))@[e_1, t_{k(1)}].$$

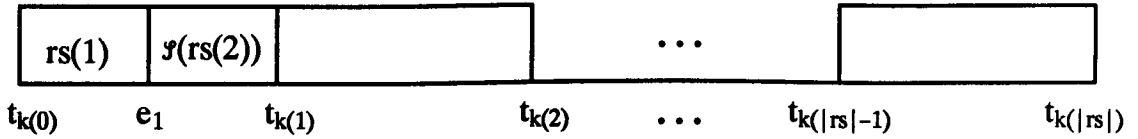


Figure 8.5. Postulated Check for Start of Sequence rs

b. There exists a sequence of $|rs|-2$ time points denoted by $e_2, \dots, e_{|rs|-1}$ such that for all i in $\{2, \dots, |rs|-1\}$ e_i is an element of $[t_{k(i-1)}, t_{k(i)}]$ and $(e_i - e_{i-1})$ is at least $LB(rs(i))$ and at most $UB(rs(i))$ and H ends $rs(i)$ during $[t_{k(i-1)}, e_i]$ and H starts $rs(i+1)$ during $[e_i, t_{k(i)}]$.

More precisely,

$$\exists e_2, \dots, e_{|rs|-1} \in T: \forall i \in \{2, \dots, |rs|-1\}: e_i \in [t_{k(i-1)}, t_{k(i)}] \wedge LB(rs(i)) \leq (e_i - e_{i-1}) \leq UB(rs(i))$$

$$H \text{ sat } \mathcal{E}(rs(i))@[t_{k(i-1)}, e_i] \wedge H \text{ sat } \mathcal{J}(rs(i+1))@[e_i, t_{k(i)}].$$

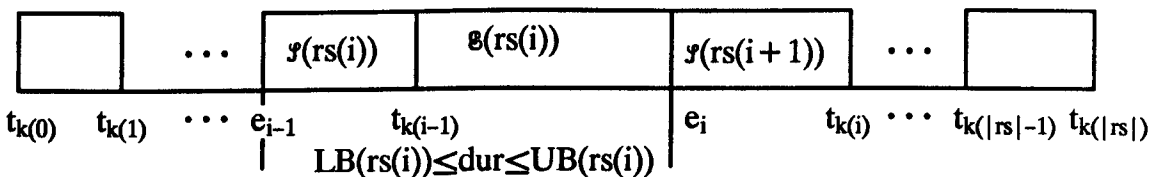


Figure 8.6. Postulated Check for a Typical Mode in Sequence rs

c. H ends $rs(|rs|)$ during $[t_{k(|rs|-1)}, t_{k(|rs|)}]$ and $(t_{k(|rs|)} - e_{|rs|-1})$ is at least $LB(rs(|rs|))$ and at most $UB(rs(|rs|))$.

More precisely,

$H \text{ sat } \mathfrak{g}(rs(|rs|))@[t_{k(|rs|-1)}, t_{k(|rs|)}] \wedge LB(rs(|rs|)) \leq (t_{k(|rs|)} - e_{|rs|-1}) \leq UB(rs(|rs|))$.

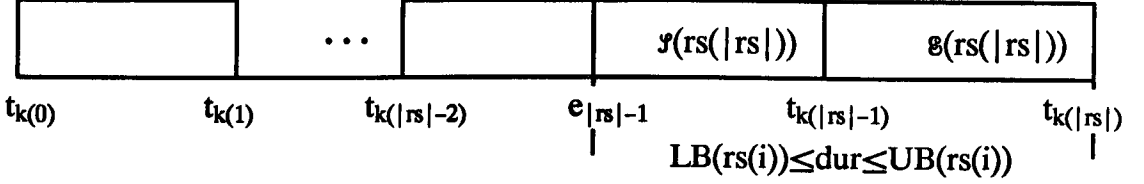


Figure 8.7. Postulated Check for End of Sequence rs

In the next section we present a set of mission verification checks which confirm the postulated checks; and then prove that the mission verification checks do indeed confirm the mission verification condition. Before presenting the mission verification checks, consider the following check which confirms the postulated check.

For any mode m for the mode set of MRS , any history H from $MEDH$, and any interval Int if H satisfies $CF(m)$ during Int then H must satisfy m during Int .

More precisely,

$\forall m \in M(MRS): \forall H \in MEDH: \forall Int \in SI(T): [H \text{ sat } CF(m)@Int \Rightarrow H \text{ sat } m@Int]$.

$MRS \text{ cmp } MEDH$.

The above check, is a simple check which allows each mode to be considered in detail. However this check is too stringent, since it does not exploit fact the fact that at the the start point of the controller graph of a mode the controller graph of a predecessor mode of that mode has ended or the possibility of allowing a delay between the end point of a mode and its controller graph.

Mission Verification Checks

In this section a set of mission verification checks derived by using the insight gained from studying the postulated checks, are presented next.

Definition: Mission verification checks

The controller function CF of a mission real world specification MRS passes the mission

verification checks if and only if there exists a precondition function $PF: M(MRS) \rightarrow PredSet$ and a delay function $\Delta f: M(MRS) \rightarrow T$ (for which $\Delta f(E(MRS)) = 0$) such that the following five conditions hold.

i. All histories of MEDH, must satisfy $PF(S(MRS))$ at $s(T)$.

$\forall H \in MEDH: H \text{ sat } PF(S(MRS))@s(T)$.

ii. For any history H from MEDH, any interval Int , if H satisfies $PF(S(MRS))$ at $s(Int)$ and $CF(S(MRS))$ during Int , then there exists a time point t during Int at most $\Delta f(S(MRS))$ before $e(Int)$ such that:

a. H satisfies $S(MRS)$ during $[s(Int), t]$; and

b. for any x a successors of $S(MRS)$ if H satisfies the start and invariant of $S(CF(x))$ at $e(Int)$, then H must start during $[t, e(Int)]$ and satisfy $PF(x)$ at $e(Int)$.

$\forall H \in MEDH: \forall Int \in SI(T):$

[$H \text{ sat } PF(S(MRS))@s(Int) \wedge H \text{ sat } CF(S(MRS))@Int \Rightarrow$

$\exists t \in Int: (e(Int) - t) \leq \Delta f(S(MRS)) \wedge$

a. $H \text{ sat } S(MRS)@[s(Int), t] \wedge$

b. $\forall x \in MRS.sr(S(MRS)): H \text{ sat } Start(S(CF(x))) \wedge Inv(S(CF(x)))@e(Int)$

$\Rightarrow H \text{ sat } \mathcal{P}(x)@[t, e(Int)] \wedge H \text{ sat } PF(x)@e(Int)]$.

iii. For any arc (x, y) of $A(MRS)$, for any history H from MEDH, any interval Int , if H satisfies $PF(x)$ at $s(Int)$ and $CF(x)$ during Int , then there exists a time point t during Int at most $\Delta f(x)$ before $e(Int)$ such that:

a. x ends during $[s(Int), t]$ for H ; and

b. if H satisfies the start and invariant of $S(CF(y))$ at $e(Int)$, then H must start y during $[t, e(Int)]$ and satisfy $PF(y)$ at $e(Int)$.

$\forall (x, y) \in A(MRS): \forall H \in MEDH: \forall Int \in SI(T):$

[$H \text{ sat } PF(x)@s(Int) \wedge H \text{ sat } CF(x)@Int \Rightarrow \exists t \in Int: (e(Int) - t) \leq \Delta f(x) \wedge$

a. $H \text{ sat } \mathcal{E}(x)@[s(Int), t] \wedge$

b. $H \text{ sat } Start(S(CF(y))) \wedge Inv(S(CF(y)))@e(Int)$

$\Rightarrow H \text{ sat } \mathcal{P}(y)@[t, e(Int)] \wedge H \text{ sat } PF(y)@e(Int)]$.

iv. For any history H from MEDH, any interval Int , if H satisfies $PF(E(MRS))$ at $s(Int)$ and $CF(E(MRS))$ during Int , then $E(MRS)$ ends during Int for H .

$\forall H \in MEDH: \forall Int \in SI(T): H \text{ sat } PF(E(MRS))@s(Int) \wedge H \text{ sat } CF(E(MRS))@Int \Rightarrow H \text{ sat } g(E(MRS))@Int.$

v. For any mode m of MRS the lower bound of $CF(m)$ must be at least the lower bound of m plus $\Delta f(m)$ and the upper bound of $CF(m)$ must be at most the upper bound of m minus $\Delta_{max}(m)$, where $\Delta_{max}(m)$ is the maximum delay for a predecessor of m .

$\forall m \in M(MRS): [LB(CF(m)) \geq LB(m) + \Delta f(m) \wedge UB(CF(m)) \leq UB(m) - \Delta_{max}(m)],$

where $\Delta_{max}(m) = \text{maximum } \{\Delta f(x) \mid x \in MRS.pr(m)\}, \text{ if } x \neq S(MRS),$
 $0, \text{ if } x = S(MRS).$

Remark: $LB(MG) = \text{minimum } \left\{ \sum_{i=0}^{i=|cs|} LB(cs(i)) \mid cs \in Seq(MG) \right\};$ and

$UB(MG) = \text{maximum } \left\{ \sum_{i=0}^{i=|cs|} UB(cs(i)) \mid cs \in Seq(MG) \right\}.$

The fact that a precondition function PF , a controller function CF and a delay function Δf pass the mission verification checks for a mission real world specification MRS will be denoted by $\langle CF, PF, \Delta f \rangle \text{ pass MRS}.$

Mission Verification Checks Theorem

The mission verification checks theorem (theorem 8.4) shows that if these verification checks can be confirmed then the mission controller specification given by MCGT will satisfy the mission verification conditions.

Theorem 8.4

If the mission controller specification of a system passes the mission verification checks then the mission verification condition holds for that system.

More precisely,

$\langle CF, PF, \Delta f \rangle \text{ pass MRS} \Rightarrow$

$\forall H \in MEDH: H \text{ sat } MCGT(MRS, CF) \Rightarrow H \text{ sat } MRS.$

Proof.

The proof follows from lemma 4.1, lemma 4.7, lemma 8.1 and the verification checks.

From lemma 4.7, we have:

1. $H \text{ sat MCGT}(\text{MRS}, \text{CF}) \Rightarrow$

$\exists cs \in \text{MCGT}(\text{MRS}, \text{CF}): \exists t_0, \dots, t_{|cs|}: H \text{ sat } cs@ \langle t_0, \dots, t_{|cs|} \rangle \wedge t_0 = s(T) \wedge t_{|cs|} = e(T).$

From lemma 8.1, we have:

2. $\exists rs \in \text{Seq}(\text{MRS}): \exists k \in \text{KF}: \forall i \in \{1, \dots, |rs|\}: cs(k(i-1)+1, k(i)) \in \text{Seq}(\text{CF}(rs(i))).$

From the mission verification checks we make the observation that the i th mode of rs is satisfied during an interval contained within the interval $[t_{k(i-2)}, t_{k(i)}]$. Hence we can conclude that by time point $t_{k(i)}$ the first i modes of rs have been satisfied. Based on this observation we make an assumption of the form that at a time point t prior to $t_{k(i)}$ the sequence $rs(1, i)$ is satisfied and the mode $rs(i+1)$ is start satisfied during $[t, t_{k(i)}]$.

Next we show $\forall i \in \{1, \dots, |rs|-1\}: \text{PS}(i)$, where $\text{PS}(i)$ is:

$H \text{ sat } cs(1, k(i))@[t_0, t_{k(i)}] \Rightarrow$

$\exists e_i \in [t_0, t_{k(i)}]: 0 \leq (t_{k(i)} - e_i) \leq \Delta f(rs(i)) \wedge$

$H \text{ sat } rs(1, i)@[t_0, e_i] \wedge H \text{ sat } \mathcal{J}(rs(i+1))@[e_i, t_{k(i)}] \wedge H \text{ sat } \text{PF}(rs(i+1))@t_{k(i)}.$

This property is illustrated in figure 8.8.

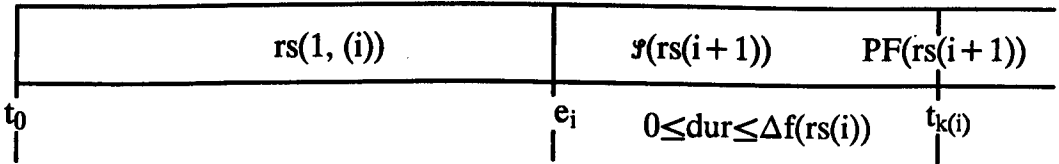


Figure 8.8. History Graph of $\text{PS}(i)$

$\text{PS}(1)$, follows from clauses i and ii of the mission verification checks.

$\text{PS}(1)$ is:

$H \text{ sat } cs(1, k(1))@[t_0, t_{k(1)}] \Rightarrow$

$\exists e_1 \in [t_0, t_{k(1)}]: 0 \leq (t_{k(1)} - e_1) \leq \Delta f(rs(1)) \wedge$

$H \text{ sat } rs(1)@[t_0, e_1] \wedge H \text{ sat } \mathcal{J}(rs(2))@[e_1, t_{k(1)}] \wedge H \text{ sat } \text{PF}(rs(2))@t_{k(1)}.$

From 2, $cs(1, k(1)) \in \text{Seq}(\text{CF}(rs(1)))$ and since MRS is an SEMG $rs(1) = S(\text{MRS})$

$\therefore H \text{ sat } \text{CF}(S(\text{MRS}))@[t_0, t_{k(1)}].$

From clause i of the mission verification checks, $H \text{ sat } PF(S(MRS))@t_0$.

Therefore from clause ii (part a), we have:

$$\exists e_1 \in [t_0, t_{k(1)}]: 0 \leq (t_{k(1)} - e_1) \leq \Delta f(rs(1)) \wedge H \text{ sat } rs(1)@[t_0, e_1].$$

Since $cs(k(1) + 1, k(2)) \in Seq(CF(rs(2)))$, we have $H \text{ sat } CF(rs(2))@[t_{k(1)}, t_{k(2)}]$

$$\Rightarrow H \text{ sat } S(CF(rs(2)))@[t_{k(1)}, t_{k(1)+1}]$$

$$\Rightarrow H \text{ sat } Start(S(CF(rs(2)))) \wedge Inv(S(CF(rs(2))))@t_{k(1)}.$$

Therefore from clause ii (part b) we have: $H \text{ sat } \mathcal{F}(rs(2))@[e_1, t_{k(1)}] \wedge H \text{ sat } PF(rs(2))@t_{k(1)}$.

Next we show that assuming $PS(i)$, we can prove $PS(i + 1)$, using clauses iii and v of the mission verification checks. We show, $\forall i \in \{1, \dots, |rs|-2\}: PS(i) \Rightarrow PS(i + 1)$.

$$H \text{ sat } cs(1, k(i+1))@[t_0, t_{k(i+1)}] \Leftrightarrow$$

$$H \text{ sat } cs(1, k(i))@[t_0, t_{k(i)}] \wedge H \text{ sat } cs(k(i) + 1, k(i+1))@[t_{k(i)}, t_{k(i+1)}].$$

From 2, $cs(k(i) + 1, k(i+1)) \in Seq(CF(rs(i+1)))$.

From $PS(i)$, we have: $H \text{ sat } cs(1, k(i))@[t_0, t_{k(i)}] \Rightarrow$

$$\exists e_i \in [t_0, t_{k(i)}]: 0 \leq (t_{k(i)} - e_i) \leq \Delta f(rs(i)) \wedge$$

$$H \text{ sat } rs(1, i)@[t_0, e_i] \wedge H \text{ sat } \mathcal{F}(rs(i+1))@[e_i, t_{k(i)}] \wedge H \text{ sat } PF(rs(i+1))@t_{k(i)}.$$

Hence, from clause iii , part a of the mission verification checks with $x = rs(i + 1)$ we have:

$$\exists e_{i+1} \in [t_{k(i)}, t_{k(i+1)}]: 0 \leq (t_{k(i+1)} - e_{i+1}) \leq \Delta f(rs(i+1)) \wedge H \text{ sat } \mathcal{G}(rs(i+1))@[t_{k(i)}, e_{i+1}].$$

Since $cs(k(i+1) + 1, k(i+2)) \in CF(rs(i+2))$ we have $H \text{ sat } CF(rs(i+2))@[t_{k(i+1)}, t_{k(i+2)}]$.

$$\Rightarrow H \text{ sat } S(CF(rs(i+2)))@[t_{k(i+1)}, t_{k(i+1)+1}].$$

$$\Rightarrow H \text{ sat } Start(S(CF(rs(i+2)))) \wedge Inv(S(CF(rs(i+2))))@t_{k(i+1)}.$$

Hence, from clause iii , part b of the mission verification checks with $y = rs(i + 2)$ we have:

$$H \text{ sat } \mathcal{F}(rs(i+2))@[e_{i+1}, t_{k(i+1)}] \wedge H \text{ sat } PF(rs(i+2))@t_{k(i+1)}.$$

From clause v of the mission verification check we have:

$$LB(CF(rs(i+1))) \geq LB(rs(i+1)) + \Delta f(rs(i+1)) \wedge$$

$$UB(CF(rs(i+1))) \leq UB(rs(i+1)) - \Delta_{max}(rs(i+1)).$$

In the following we denote: $LB(CF(rs(i+1)))$ by LB , $UB(CF(rs(i+1)))$ by UB , $LB(rs(i+1))$ by LB' and $UB(rs(i+1))$ by UB' .

We wish to show that: $LB' \leq (e_{i+1} - e_i) \leq UB'$.

$$\text{We have: } 0 \leq (t_{k(i+1)} - e_{i+1}) \leq \Delta f(rs(i+1)) \quad 3.$$

$$\text{and } 0 \leq (t_{k(i)} - e_i) \leq \Delta f(rs(i)) \quad 4.$$

By subtracting 3 from 4:

$$-\Delta f(rs(i+1)) \leq (t_{k(i)} - e_i) - (t_{k(i+1)} - e_{i+1}) \leq \Delta f(rs(i)).$$

Which can be rewritten as:

$$-\Delta f(rs(i+1)) \leq t_{k(i)} - t_{k(i+1)} + e_{i+1} - e_i \leq \Delta f(rs(i)).$$

By subtracting $t_{k(i)}$ and adding $t_{k(i+1)}$ we have:

$$(t_{k(i+1)} - t_{k(i)}) - \Delta f(rs(i+1)) \leq e_{i+1} - e_i \leq (t_{k(i+1)} - t_{k(i)}) + \Delta f(rs(i)).$$

Using $LB \leq (t_{k(i+1)} - t_{k(i)}) \leq UB$, we get:

$$LB - \Delta f(rs(i+1)) \leq (e_{i+1} - e_i) \leq UB + \Delta f(rs(i)).$$

From the definition of Δ_{\max} we can infer:

$$LB - \Delta f(rs(i+1)) \leq (e_{i+1} - e_i) \leq UB + \Delta_{\max}(rs(i+1)).$$

From clause ν of the mission verification checks we can infer the following:

$$LB - \Delta f(rs(i+1)) \geq LB' \text{ and } UB + \Delta_{\max}(rs(i+1)) \leq UB'.$$

$$\therefore LB' \leq (e_{i+1} - e_i) \leq UB'.$$

This situation is illustrated in figure 8.9

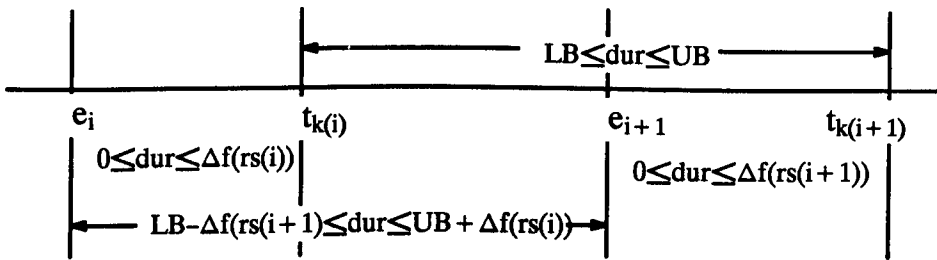


Figure 8.9. Time bounds and Delay Function

Therefore, from lemma 4.1, we can state:

$$H \text{ sat } \mathcal{J}(rs(i+1))@[e_i, t_{k(i)}] \wedge H \text{ sat } \mathcal{G}(rs(i+1))@[t_{k(i)}, e_{i+1}] \wedge LB' \leq (e_{i+1} - e_i) \leq UB'$$

$$\Rightarrow H \text{ sat } rs(i+1)@[e_i, e_{i+1}].$$

$$H \text{ sat } rs(1, i)@[t_0, e_i] \wedge H \text{ sat } rs(i)@[e_i, e_{i+1}] \Rightarrow H \text{ sat } rs(1, i+1)@[t_0, e_{i+1}].$$

$$\therefore PS(i+1).$$

From $PS(|rs|-1)$ we have: $\exists e_{|rs|-1} \in [t_0, t_{k(|rs|-1)}]: 0 \leq (t_{k(|rs|-1)} - e_{|rs|-1}) \leq \Delta f(rs(|rs|-1)) \wedge$
 $H \text{ sat } rs(1, k(|rs|-1))@[t_0, e_{|rs|-1}] \wedge H \text{ sat } \gamma(rs(|rs|))@[e_{|rs|-1}, t_{k(|rs|)}] \wedge$
 $H \text{ sat } PF(rs(|rs|))@t_{k(|rs|)}.$

From *clause iv* of the mission verification checks, $rs(|rs|) = E(MRS)$ and $cs(k(|rs|-1) + 1, k(|rs|)) \in Seq(CF(rs(|rs|)),$ we have: $H \text{ sat } g(rs(|rs|))@[t_{k(|rs|-1)}, t_{k(|rs|)}].$

Next we show $LB(rs(|rs|)) \leq (t_{k(|rs|)} - e_{|rs|-1}) \leq UB(rs(|rs|)).$

Since $\Delta f(E(MRS)) = 0$ we show: $LB(CF(rs(|rs|))) \leq (t_{k(|rs|)} - e_{|rs|-1}) \leq UB(rs(|rs|)).$

From the definition of Δ_{max} , we have: $0 \leq (t_{k(|rs|-1)} - e_{|rs|-1}) \leq \Delta_{max}(rs(|rs|)).$

$$\begin{aligned} (t_{k(|rs|)} - e_{|rs|-1}) &\leq t_{k(|rs|)} - (t_{k(|rs|-1)} - \Delta_{max}(rs(|rs|))) \\ &\leq t_{k(|rs|)} - t_{k(|rs|-1)} + \Delta_{max}(rs(|rs|)) \\ &\leq UB(CF(rs(|rs|))) + \Delta_{max}(rs(|rs|)) \\ &\leq UB(rs(|rs|)). \end{aligned}$$

$$\begin{aligned} (t_{k(|rs|)} - e_{|rs|-1}) &\geq t_{k(|rs|)} - t_{k(|rs|-1)} \\ &\geq LB(CF(rs(|rs|))). \end{aligned}$$

Hence, we have: $H \text{ sat } rs(|rs|)@[e_{|rs|-1}, t_{k(|rs|)}].$

$\therefore H \text{ sat } rs(1, |rs|)@[t_0, t_{k(|rs|)}], k(|rs|) = |cs|, t_{k(|rs|)} = e(T)$, hence $H \text{ sat } rs$.

Mission Production Rules

In this section I will discuss the production rules that can be derived from the mission verification checks; and their applicability for the development strategy.

Rule a. This production rule ensures that *clause v* holds for a mode m .

The lower bound of the $CF(m)$ is at least the lower bound of m plus the delay of m ; and the upper bound of $CF(m)$ is at most the upper bound of m minus the maximum delay of m .
 $LB(CF(m)) \geq LB(m) + \Delta f(m) \wedge UB(CF(m)) \leq UB(m) - \Delta_{max}(m).$

Remark: To check the above rule for a mode m the delay function must be specified for all predecessors of m .

Rule b'. This is simply *clause ii*.

Rule c'. This construction rule ensures that *clause iii* holds for all arcs in which m is the first mode.

For any history H from $MEDH$, any interval Int if H satisfies $PF(m)$ at $s(Int)$ and $CF(m)$ during Int then there exists a time point t during Int at most $\Delta f(m)$ before $e(Int)$ such that m ends during $[s(Int), t]$; and for any successor x of m if the start and invariant of $S(CF(x))$ are satisfied at $e(Int)$ then x starts during $[t, e(Int)]$ and the precondition of x is satisfied at $e(Int)$.

$\forall H \in MEDH: \forall Int \in SI(T):$

$[(H \text{ sat } PF(m)@s(Int) \wedge H \text{ sat } CF(m)@Int \Rightarrow \exists t \in Int: (e(Int) - t) \leq \Delta f(m)$

$H \text{ sat } g(m)@[s(Int), t]) \wedge$

$(\forall x \in MRS.sr(m): H \text{ sat } Start(S(CF(x))) \wedge Inv(S(CF(x)))@e(Int) \Rightarrow$

$H \text{ sat } p(x)@[t, e(Int)] \wedge H \text{ sat } PF(x)@e(Int))].$

Rule d. This is simply clause *iv*.

To check the rule b' and c' for a mode m the precondition and controller function must be specified for all successors of m . The dependency of the checks on the controller function of other modes, reduces the usefulness of the check during the production of the mission controller specification. To allow a similar check to be performed after the analysis of each mode the notion of a template function is introduced, as a function that allows the mission controller analysts to specify the state of the mission controller at the start of a controller graph. A template function is used to define rules b , c and e which are based on rules b' and c' .

Definition: template function

The template function of a mission real world specification is a function from the set of modes of the mission real world specification to a set of system predicates imposed over the mission controller variables.

TF: $M(MRS) \rightarrow PredSet$, where $PredSet$ is a set of system predicates over the mission controller variables.

Rule c. This rule together with the rule e confirms that clause *ii* holds.

For any history H from $MEDH$ and any interval Int , if H satisfies $PF(S(MRS))$ at $s(Int)$ and $CF(S(MRS))$ during Int then there exists a time point t during Int at most $\Delta f(S(MRS))$

before $e(Int)$ such that m ends during $[s(Int), t]$; and for any successor x of m for which $TF(x)$ is satisfied at $e(Int)$, x starts during $[t, e(Int)]$ and $PF(x)$ is satisfied at $e(Int)$.

$\forall H \in MEDH: \forall Int \in SI(T):$

$[H \text{ sat } PF(S(MRS))@s(Int) \wedge H \text{ sat } CF(S(MRS))@Int \Rightarrow \exists t \in Int: (e(Int) - t) \leq \Delta f(S(MRS))$
 $H \text{ sat } g(S(MRS))@[s(Int), t] \wedge$
 $(\forall x \in MRS.sr(S(MRS)): H \text{ sat } TF(x)@e(Int) \Rightarrow H \text{ sat } f(m)@[t, e(Int)] \wedge H \text{ sat } PF(x)@e(Int))].$

Rule c. This rule together with the rule *e* confirms that clause *iii* holds for the arcs that start with mode m .

For any history H from $MEDH$ and any interval Int , if H satisfies $PF(m)$ at $s(Int)$ and $CF(m)$ during Int then there exists a time point t during Int at most $\Delta f(m)$ before $e(Int)$ such that m ends during $[s(Int), t]$; and for any successor x of m for which $TF(x)$ is satisfied at $e(Int)$, x starts during $[t, e(Int)]$ and $PF(x)$ is satisfied at $e(Int)$.

$\forall H \in MEDH: \forall Int \in SI(T):$

$[H \text{ sat } PF(m)@s(Int) \wedge H \text{ sat } CF(m)@Int \Rightarrow \exists t \in Int: (e(Int) - t) \leq \Delta f(m)$
 $H \text{ sat } g(m)@[s(Int), t] \wedge$
 $(\forall x \in MRS.sr(m): H \text{ sat } TF(x)@e(Int) \Rightarrow H \text{ sat } f(m)@[t, e(Int)] \wedge H \text{ sat } PF(x)@e(Int))].$

Rule e.

The conjunction of the start and invariant predicate of $S(CF(m))$ imply $TF(m)$ under the normal histories.

$\forall H \in MEDH: \forall t \in T: [H \text{ sat } Start(S(CF(m))) \wedge Inv(S(CF(m)))@T \Rightarrow H \text{ sat } TF(m)@t].$

Lemma 8.2

If production rules b , c and e are confirmed for all the modes then production rules b' and c' must hold for all the modes.

Proof. This result follows simply from the fact that we can substitute $Start(CF(m)) \wedge Inv(CF(m))$ for $TF(x)$, since $\forall H \in MEDH: \forall t \in T: [H \text{ sat } Start(S(CF(x))) \wedge Inv(S(CF(x)))@t \Rightarrow H \text{ sat } TF(x)@t]$ (using the transitive property of \Rightarrow).

8.5.2. Mission Controller Development Strategy

The development strategy follows the three stages shown in figure 8.2. In the first stage an outline of the mission controller specification is constructed by defining the precondition function, template function and delay function of the mission real world specification. In the second stage, the controller function is constructed. Finally, in the third stage, the completeness and consistency of the mission controller specification is checked and the transformation MCGT used to construct the mission controller specification.

Construct Outline Specification

Step 1 (Precondition Function)

Construct a *precondition function* PF for MRS. As an initial guide the precondition of a mode m of MRS should be defined as: $PF(m) = \text{Start}(m) \wedge \text{Inv}(m)$.

Step 2 (Template Function)

Construct a *template function* TF for MRS. The template predicate of a mode m of MRS is defined by a systematic analysis of $PF(m)$ and the sensors and actuators of the mission controller. This systematic analysis consists of the following three steps.

- a. Make a list (PV) of all the variables over which $PF(m)$ is imposed.
- b. Make a list (CV) of all sensors and actuators of the mission controller that are related to the variables in PV; and the relevant relations (RV). These variables and relations can be identified by looking up the related variables column in the relations table of MED.
- c. Identify the constraints (by inspecting the relations in RV) which must be imposed over the variables in the list CV for the condition defined by $PF(m)$ to be maintained.

Step 3 (Delay Function)

Construct the *delay function* Δf for MRS. A full definition of the delay function of MRS can be given only after the controller function has been defined. In this step a partial definition is given, by identifying those mode with a delay of zero; and assigning variables to those mode with non-zero delays. The delay function of a mode is defined as zero if the end predicate of the mode involves a state variable of the operator console, and the delay function of the end mode is zero (i.e. $\Delta f(E(MRS)) = 0$).

Construct controller function

In this section, a set of guidelines for the construction of a *controller function* for the MRS of the system under analysis, is given. The guidelines are defined in two steps. Firstly, the controller graph of a mode m is constructed by a systematic investigation of the behaviour defined by m . This mode mode graph which defines constraints over the sensors and actuators of the mission controller and the variables of the operator console. Secondly, this mode graph is checked against the production rules for m .

Step 1 (Controller graph)

- a. Identify the sensors and actuators of the mission controller that are related to the system predicates of the mode m . These sensors and actuators can be identified by simply checking the related variables column of the description relation table of MED. Make a list (denoted by MV) of all the related sensors and actuators and a list of the appropriate relations (RV).
- b. Informally, state the basic strategy that will be used to perform the task specified by m , by manipulating the variables on the list MV.
- c. Identify the tasks (that must be performed by the mission controller) which must be specified to formulate the strategy; and the relationships between these tasks. Check that the invariant predicate of m will hold during the tasks.
- d. Identify the invariant of $CF(m)$, by checking the list MV to determine those variables that can be constrained by an invariant during $CF(m)$.
- e. By analysing the informal specification of the tasks construct modes that specify the tasks (and state informal specifications for each mode). Then use these modes to construct $CF(m)$.
- f. Check the completeness and consistency of $CF(m)$.
- g. If m is the start mode of MRS, then the following condition must be checked (it confirms clause i of the mission verification checks):

$$\forall H \in MEDH: H \text{ sat } PF(m)@s(T) \wedge H \text{ sat } \mathcal{I}(m)@s(T).$$

Step 2 (Verification checks)

In this step $CF(m)$ constructed in *step 1* is checked against production rules a and e . Then if m is S(MRS) check $CF(m)$ against rule b , if m is E(MRS) check $CF(m)$ against rule d , otherwise check $CF(m)$ against rule c . If $CF(m)$ passes the relevant rules, the analysts proceed to the next mode, otherwise $CF(m)$ must be modified. Possible modifications are suggested in *step 3*.

Step 1 and *step 2* must be repeated until the controller function is defined for all the modes.

Step 3 (Modifications)

This step must be performed if the controller graph of a mode m does not satisfy a production rules. Four options are outlined below, in the order that they should be attempted.

Modify controller graph specification. This should be the first option attempted since any changes are restricted to the mode graph of m .

Modify outline specification. The analysts must modify the precondition function, template predicate or delay function. The main drawback with this option is, that some of the production rules of the previously checked controller graphs must be checked again.

Modify environment description. Unlike, the first three options, the mission controller analysts do not have direct control over the relationships of the MED (i.e., they can only specify the properties of the sensors and actuators that are provided). The mission controller analyst should identify the relations that would be required for m to pass the production rule. If the analysts are fortunate these additional relations may hold for the sensors or actuators, otherwise modifications to the sensors or actuators or changes to the physical process would be required. Modifications to the MED, can lead to the need to check rules of modes other than m .

Modify mission real world specification. As for the the third option, the controller analysts cannot modify the MRS. The analysts can suggest changes to the mission analysts (who should then consult the customer).

If for a particular case, a simple fix can be seen by any of the options given above then it should of course be performed directly.

Mode Graph Checks

These guidelines are given in three steps. The first two steps concern the completeness and consistency checks of the mission controller specification; and the fourth the combination of the graphs defined by the controller function

Step 1 (Completeness)

A check for the completeness of the mission controller specification obtained by applying the function *MCGT* to the controller function *CF* of a mission real world specification *MRS* is given below.

External completeness check

For any mode m of *MRS*, if any history H of *MEDH* satisfies $E(CF(m))$ during any interval Int there exists a mode x (a successor of m) such that H satisfies the start and invariant predicate of $S(CF(x))$ at $e(Int)$.

$$\forall m \in M(MRS): \forall H \in MEDH: \forall Int \in SI(T): \exists x \in MRS.sr(m):$$

$$[H \text{ sat } E(CF(m))@Int \Rightarrow H \text{ sat } Start(S(CF(x)) \wedge Inv(S(CF(x)))@e(Int)]$$

The above assertion will be referred to as the external completeness check, and will be denoted by $\langle MRS, CF \rangle Ecom MED$. If the completeness of the controller graphs have been confirmed, then the external completeness check is sufficient to confirm that the mode graph given by *MCGT* is complete.

Step 2 (Consistency)

A check for the consistency of the mission controller specification obtained by applying the function *MCGT* to the controller function *CF* of a mission real world specification *MRS* is given below.

External consistency check

For any arc (x, y) of *MRS*, there exists a state value V from the universal state space such that V satisfies the invariant predicate and end predicate of $E(CF(x))$, the start predicate, invariant predicate and negation of the end predicate of $S(CF(y))$; and the invariant

relations of *MED*.

$\forall (x, y) \in A(MRS): \exists V \in \Gamma:$

$[V \text{ sat } \text{Inv}(E(CF(x))) \wedge \text{End}(E(CF(x))) \wedge \text{Start}(S(CF(y))) \wedge \text{Inv}(S(CF(y))) \wedge \neg \text{End}(S(CF(y))) \wedge C(\text{IR}(\text{MED}))].$

We will refer to the above assertion as the external consistency check. It will be denoted by $\langle MRS, CF \rangle \text{Econ MED}$.

If the consistency of the controller graphs have been confirmed, then the external consistency check is sufficient to confirm that the mode graph given by MCGT is consistent.

Step 3 (Construction of Mission Controller Specification)

The mission controller specification is constructed by applying the transformation MCGT, over MRS and CF.

8.5.3. Mission Controller Specification Theorems

In this section I will present a theorem and two lemmas that show the production guidelines are sufficient to produce a mission controller specification that will satisfy the mission verification checks, that is both complete and consistent.

Mission Production Theorem

The mission production theorem (theorem 8.5) shows that if the production rules are passed then CF, PF and Δf pass the mission verification checks.

Theorem 8.5.

If each controller graph passes the relevant production rules for a mission real world specification then the controller function CF, precondition function PF and delay function Δf pass the mission verification checks.

Proof.

This result is proven by showing how each clause of the mission verification checks is satisfied.

Clause *i*.

$$\forall H \in \text{MEDH}: H \text{ sat } \text{PF}(\text{S}(\text{MRS}))@s(\text{T}) \wedge H \text{ sat } \text{f}(\text{S}(\text{MRS}))@s(\text{T}).$$

Follows directly from *step 1.g*.

Clause *ii*.

$$\forall H \in \text{MEDH}: \forall \text{Int} \in \text{SI}(\text{T}):$$

$$[H \text{ sat } \text{PF}(\text{S}(\text{MRS}))@s(\text{Int}) \wedge H \text{ sat } \text{CF}(\text{S}(\text{MRS}))@s(\text{Int}) \Rightarrow \exists t \in \text{Int}: (e(\text{Int}) - t) \leq \Delta f(\text{S}(\text{MRS}))$$

$$\wedge H \text{ sat } \text{S}(\text{MRS})@[s(\text{Int}), t] \wedge$$

$$\forall x \in \text{MRS.sr}(\text{S}(\text{MRS})): H \text{ sat } \text{Start}(\text{S}(\text{CF}(x))) \wedge \text{Inv}(\text{S}(\text{CF}(x)))@e(\text{Int})$$

$$\Rightarrow H \text{ sat } \text{f}(x)@[t, e(\text{Int})] \wedge H \text{ sat } \text{PF}(x)@e(\text{Int})].$$

From rules *b* and *e* we can infer that rule *b'* holds for all *m* of MRS (see lemma 8.2):

Clause *iii*.

$$\forall (x, y) \in \text{A}(\text{MRS}): \forall H \in \text{MEDH}: \forall \text{Int} \in \text{SI}(\text{T}):$$

$$[H \text{ sat } \text{PF}(x)@s(\text{Int}) \wedge H \text{ sat } \text{CF}(y)@s(\text{Int}) \Rightarrow \exists t \in \text{Int}: (e(\text{Int}) - t) \leq \Delta f(x) \wedge$$

$$H \text{ sat } \text{g}(x)@[s(\text{Int}), t] \wedge$$

$$H \text{ sat } \text{Start}(\text{S}(\text{CF}(y))) \wedge \text{Inv}(\text{S}(\text{CF}(y)))@e(\text{Int})$$

$$\Rightarrow H \text{ sat } \text{f}(y)@[t, e(\text{Int})] \wedge H \text{ sat } \text{PF}(y)@e(\text{Int})].$$

From rules *c* and *e* we can infer that rule *c'* holds for all *m* of MRS (see lemma 8.2):

$$\forall m \in \text{M}(\text{MRS}): \forall H \in \text{MEDH}: \forall \text{Int} \in \text{SI}(\text{T}):$$

$$[H \text{ sat } \text{PF}(m)@s(\text{Int}) \wedge H \text{ sat } \text{CF}(m)@s(\text{Int}) \Rightarrow \exists t \in \text{Int}: (e(\text{Int}) - t) \leq \Delta f(m)$$

$$H \text{ sat } \text{g}(m)@[s(\text{Int}), t] \wedge$$

$$(\forall x \in \text{MRS.sr}(m): H \text{ sat } \text{TF}(x)@e(\text{Int})$$

$$\Rightarrow H \text{ sat } \text{f}(m)@[t, e(\text{Int})] \wedge H \text{ sat } \text{PF}(x)@e(\text{Int})).$$

Hence clause *iii*, follows directly from the definition of the successor function of a mode graph.

Clause *iv*.

$$\forall H \in \text{MEDH}: \forall \text{Int} \in \text{SI}(\text{T}): H \text{ sat } \text{PF}(\text{E}(\text{MRS}))@s(\text{Int}) \wedge H \text{ sat } \text{CF}(\text{E}(\text{MRS}))@s(\text{Int}) \Rightarrow$$

$$H \text{ sat } \text{g}(\text{E}(\text{MRS}))@s(\text{Int}).$$

Follows directly from rule *d*.

Clause v .

$\forall m \in M(MRS): [LB(CF(m)) \geq LB(m) + \Delta f(m) \wedge UB(CF(m)) \leq UB(m) - \Delta_{\max}(m)]$.

Follows directly from rule a .

Lemma 8.3

If all the controller graphs given by a controller function CF of a mission real world specification MRS are complete for MED and the controller function satisfies the external completeness check for MED then the graph obtained by applying $MCGT$ to MRS and CF is complete for MED .

$(\forall m \in M(MRS): CF(m) \text{ com } MED) \wedge \langle MRS, CF \rangle \text{ Econ } MED$

$\Rightarrow MCGT(MRS, CF) \text{ com } MED$.

Proof. This result follows from the fact that the first condition ensures that the non-end modes of the controller graphs are complete, and the second condition ensures that the end modes of the controller graphs are complete.

Lemma 8.4

If all the controller graphs given by a controller function CF of a mission real world specification MRS are consistent for MED and the controller function satisfies the external consistency check for MED then the graph obtained by applying $MCGT$ to MRS is consistent for MED .

$(\forall m \in M(MRS): CF(m) \text{ con } MED) \wedge \langle MRS, CF \rangle \text{ Econ } MED$

$\Rightarrow MCGT(MRS, CF) \text{ con } MED$.

Proof. This result follows from the fact that the first condition ensures that the arcs of the controller graphs satisfy the mode graph consistency checks, and the second condition ensures that the arcs that connect the controller graphs satisfy the mode graph consistency checks.

8.6. Summary and Conclusions

This chapter presented methodologies for the production of the controller specifications. The specifications are derived from the formal real world specifications

Verification conditions were stated for the specifications; and verification checks that can be used to confirm the specifications derived. The notion of *production rules* was introduced, as a means to steer the production of a specification towards the verification conditions; and to check the specifications during their production. Roughly, speaking the production rules allow checks to be performed over the modes and arcs of the controller specifications, as they are being produced.

A systematic strategy for the production of the safety controller specification of a system was presented, this strategy was based on the general (phase) structure for the behaviour of a safety controller (discussed in chapter 6). For each phase a mode graph is constructed, these mode graph were checked against five production rules. The safety controller specification is then produced by the connection of these mode graphs, as defined by the function CSCS. A theorem (theorem 8.2) that shows a safety controller specification produced by following the guidelines will satisfy the safety verification condition is presented.

The strategy for the production of the mission controller specification of a system was given in two stages. In the first stage an outline of the mission controller specification of a system is produced by defining the precondition function, template function and delay function of the MRS of that system. In the second stage a mode graph is constructed for each mode of the MRS, these mode graphs are checked using three production rules. The mission controller specification is then produced by the connection of these mode graphs, as defined by the function MCGT. A theorem (theorem 8.5) that shows a mission controller specification produced by following the guidelines will satisfy the mission verification condition is presented.

The methodologies, described in this chapter, give a formal treatment of the environmental, logical and timing issues of a system. The specifications produced by following the guidelines comply with the real world specifications. However, these specifications will probably require further refinements, since the guidelines do not take into account issues related to maintainability, reliability of components, cost-effectiveness

and ease of operation. In particular, the analysis should be extended to consider faults in the safety controller. This should include faults in the actuators and sensors which result in their relationship to the real world variables deviating from that described by the relations of SED, and faults in the control system which results in the behaviour of the controller variables deviating from that specified by the safety controller specification. Despite these drawbacks the specifications provide an useful basis for subsequent controller specifications.

The main benefits from the general strategy are that the analysis required to produce the specifications follows a systematic approach, and production rules allow a proof to be constructed in parallel with the production of the specification. The systematic approach enhances the confidence that can be placed in the analysis; and the production rules improve the confidence that can be placed in the formal constructs produced by the analysis. The benefits gained from establishing a distinction between the safety and mission issues during the real world analysis are reflected during the controller analysis, since a proof of the adequacy of the the safety controller specification can be obtained without considering the behaviour of the mission controller. A further benefit of this distinction is that maintenance of the mission controller of a system (provided it does not change the SED) will not affect the safety of the system.

Chapter 9 - Summary and Conclusions

In this chapter I will summarize the material that has been presented in this thesis, discuss some of the conclusions that can be drawn from the material and indicate some of the possible areas of future work.

9.1. Thesis Summary

The general class of systems considered in the thesis are *process control systems* [Smit72]; typical application areas include chemical plants and avionic systems. The main reasons for concentrating on such systems are that i) process control systems have significant properties in common, in particular, a process control system can be modelled as three interacting components: operator, controller and physical process; and ii) many safety-critical applications are in the area of process control systems. The key component, of a process control system, is the controller; this is constructed to control the behaviour of the physical process. The controller consists of four main components: operator console, control system, sensors and actuators.

This thesis has concentrated on the requirements analysis of safety-critical computing systems. Requirements analysis plays a vital role in the development of systems, since any errors in the identified requirements will corrupt the subsequent stages of system development. Experience in safety-critical systems has shown that errors in the formulation of requirements can and do cause accidents [Leve86, Jaff89]. Current methods for requirements elicitation are based on a combination of system safety and software development techniques. A common approach is the use of *HAZOPS* (Hazard and Operability Studies) and *FTA* (Fault Tree Analysis) to identify hazards from which the software safety requirements are produced [Leve89]. General guidelines for the integration of system safety and software development techniques are available [HSE87]. However, a framework which enables and guides a formal analysis of the system from the real world issues through to the system requirements is not available [Mose90].

It was asserted that to gain a complete understanding of a safety-critical computing system, the requirements of the overall system and the properties of the environment should be analysed in a common formal framework. The benefits of using formal methods during requirements analysis include unambiguity, checks for completeness and consistency, formal verification, and the potential for using automated aids [Jaff89; Roan86]. A unified framework is needed for the analysis because safety is a global issue that can only be addressed by analysing the consequences of system behaviour in the context of the environment [Gors86; Leve86]. The benefits gained from the adoption of a common framework include: i) improved and precise communication between the members of the analysis team; ii) the ability to perform a rigorous assessment of the effect of the inherent properties of the environment on system behaviour; and iii) the ability to assess the impact of the system on the environment.

The essential attributes that an appropriate formal model must possess are that it must be able to express *physical laws*, *parallelism* and *timing issues* in a *coherent (structured) form*. The necessity for the model to be able to express physical laws stems from the fact that the behaviour of the environment is governed by such laws. The necessity to treat parallelism explicitly in specifications which include the environment has been argued by some researchers [Gors88]. Timing issues will arise in all of the stages of requirements analysis. Timing issues are present in the description of the environment, since physical laws make an explicit reference to time, and in many cases the relationships between the sensors, actuators and the physical process are dependent on time. A structured model is necessary to handle the complexity of these systems. The structure should be present in the techniques used to compose the basic constructs of the model and in the basic constructs themselves.

The approach to requirements analysis, adopted here is concerned with system behaviour at two distinct levels of abstraction: *real world* and *controller*. At the real world level it addresses the behaviour exhibited by the physical process. At the controller level, it is concerned with the behaviour that must be exhibited by the sensors and actuators of the controller for the physical process to behave as required.

For safety-critical systems it has been suggested that, to obtain a clear analysis of the safety-related properties, a distinction must be made between the safety-critical and mission-oriented behaviour of a system [Leve84; Mula86]. In this thesis, the distinction between the safety and mission issues is established during requirements analysis, by partitioning the analysis at each level into safety and mission analyses. This distinction is reflected in a general structure for safety-critical systems which decomposes the components of a process control system. This structure is realized by partitioning the operator function into the safety and mission operator, the controller into the safety and mission controller, and by defining two viewpoints over the physical process: one for the safety-critical behaviour and the other for the mission-oriented behaviour. A system development model that reflects the decisions to perform the analysis at two levels and maintain a distinction between the safety and mission issues is presented.

At the real world level, the safety analysis is performed in three stages which produces four essential specifications. In the first stage the potential *disasters* associated with the mission of the system or the environment are identified. The second stage involves the identification of the *hazards* which can lead to potential disasters. It also involves the specification of the behaviour of the environment which impinges on the safety-critical behaviour of the system by producing the *safety real world description (SRD)*. The safety real world description, and the assumption that the identified hazards specify all the conditions under which a disaster can occur, comprise the safety (real world) assumptions of the system. The third stage involves the construction of a *safety real world specification (SRS)* which specifies a behaviour that, if maintained by the system, will ensure that no *identified* disaster can occur – under the safety assumptions of the system.

At the real world level, the mission analysis is performed in two stages which produces two essential specifications. In the first stage the *mission real world specification (MRS)* is produced which specifies the mission oriented behaviour of the system. The second stage involves specifying the behaviour of the environment which impinges on the mission-oriented behaviour of the system by producing the *mission real world description (MRD)*.

At the controller level, the safety analysis is performed in two stages which produce two essential specifications. In the first stage the relationship between the *sensors and actuators* of the safety controller and the *physical process* must be specified in the formal framework. This specification together, with the SRD is referred to as the *safety environment description (SED)*. In the second stage, the behaviour that must be exhibited by the sensors and actuators and at the operator console to ensure that the behaviour complies with the safety real world specification, for the histories obtained from SED, is specified as the *safety controller specification (SCS)*. This specification is produced by an analysis of the SED and SRS.

At the controller level, the situation for the mission analysis is the same as that for the safety analysis, just replace safety by mission. This analysis leads to the production of the *mission environment description (MED)* and the *mission controller specification (MCS)*.

In chapter three, the thesis presented a specification model for requirements analysis. The semantics of the model were defined in terms of formal *histories* (functions from the system lifetime to the state space). Three sorts of relations were introduced to specify properties that are invariant over histories: *class relations*, *invariant relations* and *history relations*. To describe the restrictions imposed on system behaviour from the environment the concept of a *history description* was introduced, this is a construct which is used to specify a set of histories that satisfy the properties described by the relations of the description.

To specify the required behaviour of a system a set of satisfaction conditions was introduced, it is shown how these conditions could be used to specify the behaviour of a system at a *time point*, during an *interval* and formally captures the notion of an *event* as a “temporal marker”. Several examples of system behaviour expressed using the conditions were presented. It was concluded that though the expressive power of the conditions was sufficient (for the systems to be studied), a drawback was that the undisciplined use of the conditions could easily lead to badly structured and unreadable specifications. Two possibilities were considered to overcome this potential drawback, i)

develop a set of guidelines for a disciplined use of the conditions and ii) devise a higher-level construct based on the conditions which can be used to structure the specifications. This thesis followed the second route.

In chapter four, the structured constructs of the formal model were presented. The main construct is that of a *mode*. A mode of a system is used to specify a task of the system; it specifies the behaviour of the system at the start of a task, during the task and the behaviour which must be exhibited by the system for the task to be completed. The key feature of a mode is that it can be used to specify system *state* during a task, the *event* that marks the completion of a task, and the *timing* constraints over a task, in a coherent format. A graphical notation, a *mode graph*, was introduced to specify the transitions between the modes of a system. An informal dual of mode graphs, *phase graphs*, and a restricted class of mode graphs *single entry exit mode graphs (SEMG)* were also introduced.

Though the expressiveness of formal models has received a lot of attention from researchers, the relationship between a formal model and a production methodology has rarely been extensively examined [Wing90] (a detailed study has been performed for a structured method [Hat188]). However, a recent survey of formal methods [Stru89] concludes that most so-called formal methods are not formal methods at all, but merely formal languages or notations. The thesis took the view that for a formal model to be useful during requirements analysis, a set of systematic guidelines for the production of the *essential specifications* must be provided.

In chapter five, a relationship between the formal model and the essential specifications of the real world analysis was identified, by discussing the formalization of the essential specifications. For the safety analysis, the disasters, hazards and safety real world specification were formulated as system predicates, and the safety real world description formulated as a history description. For the mission analysis, the mission real world description is formulated as a history description. To structure the development of the mission real world specification an intermediate (less formal) specification called the

mission phase specification is introduced – this is expressed as a phase graph. The mission real world specification is formulated as an SEMG.

The safety-critical behaviour of a system SY at the end of the real world analysis is described by the pair $\langle \text{SRD}(\text{SY}), \text{SRS}(\text{SY}) \rangle$ and the mission-oriented behaviour by the pair $\langle \text{MRD}(\text{SY}), \text{MRS}(\text{SY}) \rangle$.

In chapter six, a relationship between the formal model and the essential specifications of the controller analysis was identified, by discussing the formalization of the essential specifications. For the safety analysis, the safety environment description is formulated as a history description and the safety controller specification is formulated as an SEMG. To guide the analysis of the safety controller, a general structure is presented in terms of five phases: *start up*, *monitor*, *recovery*, *reset*, *shut down* and *end*. The structures which will be used to specify the phases were described in detail. For the mission analysis, the mission environment description is formulated as a history description and the mission controller specification formulated as an SEMG. The structure of the mission controller specification is based on the mission real world specification. A general strategy for the production of the mission controller specification from the mission real world specification by the construction of SEMGs from the modes of the mission real world specification is briefly discussed.

The safety-critical behaviour of a system SY at the end of the controller analysis is described by the pair $\langle \text{SED}(\text{SY}), \text{SCS}(\text{SY}) \rangle$ and the mission-oriented behaviour by the pair $\langle \text{MED}(\text{SY}), \text{MCS}(\text{SY}) \rangle$.

The realisation of an explicit relationship between the specifications produced during requirements analysis and the formal model, provides a useful basis for development. A brief overview of the work presented in the first six chapters will soon be published [Sae90]. However, to ensure that the development proceeds in a systematic way which best utilizes the support provided by the development model and formal model, systematic step-by-step guidelines are required. Methodologies that describe such guidelines are presented in chapters seven and eight.

Chapter seven introduced some guidelines for the development of the essential specifications of the real world analysis. The real world analysis was presented in three main stages, a preliminary analysis stage, safety real world analysis and mission real world analysis. The strategy for the safety real world analysis consisted of a set of (iterative) systematic guidelines for the identification of the disaster set, hazard specification, safety real world description and safety real world specification. Validation guidelines, based on independent analysis were also presented. The strategy for the mission real world analysis consisted of a set of systematic guidelines for the development of the mission phase specification, mission real world specification and mission real world description. Guidelines for the verification of the completeness and consistency of the mission real world specification and validation of the mission real world specification against the system concept were also outlined. These guidelines exploit the modular development of the specifications.

The separation of safety and mission issues simplified the guidelines of both the safety and mission specifications. The safety analysis is simplified by focusing only on the potential disasters and their hazards; the mission analysis is simplified by removing the necessity to consider the safety-critical issues in combination with the mission.

Chapter eight presented methodologies for the development of the essential specifications of the controller analysis. It was shown how these specifications are derived from the formal specifications produced during the real world analysis, hence the potential for formal verification exists between the real world specifications and the controller specifications. Verification conditions were presented for the safety and mission controller specifications. The notion of *production rules* was introduced to steer the specifications produced by following *production guidelines* towards the verification conditions.

Overall this thesis has highlighted requirements analysis as an important (but neglected) phase of system development. It has been suggested that for the analysis to be performed in a systematic way a general structure of the system being specified should be available, and the role of the essential specifications produced during requirements analysis clearly defined. A general strategy was suggested in which a distinction is made

between the specifications at the real world level and controller level, and at each level between the safety-critical and mission-oriented issues. Furthermore this thesis has shown that formal methods can play an important role during requirements analysis, in the representation of the essential specifications and perhaps more significantly in providing a systematic approach to the production of these specifications.

9.2. Evaluation and Conclusions

The implicit representation of *time* in the semantics of the formal model enables a precise specification of timing constraints to be given and provides a means of relating the timing constraints at the controller level to the behaviour exhibited at the real world level. The encapsulation of the behaviour of the *environment* and *system* in the same framework, allows a rigorous assessment of the effect of the inherent properties of the environment on system behaviour to be performed and enables the impact of the system on the environment to be assessed.

The development model clarifies the roles that the members of the development team play during requirements analysis, thereby leading to improved and precise communication within the team.

The two level approach leads to a full understanding of system behaviour at the real world level, before any complexities are introduced from the controller; this is particularly relevant for the safety requirements since a “solution” independent statement of safety is developed. The separation of safety and mission issues allows the formal analysis to be targeted at the safety-critical issues of the system. A drawback to the use of formal methods in complex systems is that during a complex formal analysis it is possible to lose sight of the realities of the system being analysed. The explicit relationship between the formal and development models helps to keep track of the formal analysis and the properties of the system. A further benefit of identifying an explicit relationship between the specification and development models is that it allows the identification of general verification and development strategies.

The methodologies presented in chapter seven steer the development process through several intermediates stages, the documents (tables) produced at the intermediates stages act as “diaries” of the development. These diaries can be used in the certification of the development process; the diaries are also useful in the verifications of the different specifications. For the controller level specification the use of production rules allows the construction of proofs for the verification of the controller specifications in parallel with their production.

Six basic principles for a requirements analysis scheme for a safety-critical (process control) system are summarised below.

Separation. The safety and mission issues of the system should be separated during the early stages of the analysis, and the distinction maintained during the analysis. However, it is usually impossible to maintain a complete dichotomy between the mission and the critical requirements, because it would be futile to impose critical requirements that are so stringent that the system could not satisfy its mission. Some aspects of the mission requirements must be considered in the analysis of the critical requirements, to ensure that the behaviour of the safety controller will (under fault-free circumstances) permit the satisfaction of the mission requirements as well as ensuring the absence of system hazards. The analysis should be performed at two levels, the real world and controller levels. A distinction between the restrictions imposed on system behaviour (by the environment) and the required system behaviour should be made at both levels.

Simplicity. The safety controller and its interfaces should be kept as simple as possible.

Systematic. The analysis should follow a systematic step-by-step approach, which can be monitored. Verification conditions should be provided for the controller specifications; and production rules devised which steer the production of the controller specifications towards the verification conditions.

Roles. The members of the development team must be assigned clearly defined roles. In particular, those responsible for the safety analysis should be known to all members of the development team.

Targeted. Complete formal analysis should be carefully targeted to the safety-critical issues

of the system. Limited formal analysis should be performed over the mission-oriented issues.

Fault tolerance. More than one mechanism should be included in the (safety) controller to avoid a potential hazard.

9.3. Future Work

This thesis has concentrated on how a development model, formal model and a related development methodology can be used to improve requirements analysis. It is not claimed that the approach presented here is the best solution to the requirements analysis of all safety-critical systems, rather that it is a useful approach for an important class of safety-critical systems. However, the basic principles that underpin the approach could be used to provide a more complete approach for requirements analysis. Essentially, the work set out in this thesis should be considered as experimental work which provides a basis for further investigation, as opposed to a self-contained solution to the problem of requirements analysis.

The best way to check the usefulness of the framework for requirements analysis presented here is to perform worked examples. A worked example (of a simple safety-critical system) was presented which illustrated how the essential specifications can be expressed in the formal model. In appendices B and C the requirements analysis of a simple chemical plant is presented, which illustrates how the framework supports the analysis of a safety-critical system. However, for a more thorough test of the usefulness of the framework a large system must be analysed. An attractive candidate is a model railway system which the Computing Laboratory has recently acquired for experimental work on real-time systems (some preliminary work, has been performed, and the results seem encouraging). It remains to be seen how useful the framework can be for large complex systems.

After an evaluation of the proposed framework there are two possible approaches to future work. In the first approach, the framework would be extended to consider important issues in the development of safety-critical systems that were not explored by this thesis.

The second approach would be to apply the basic principles behind the framework to different formalisms. Both approaches are considered below, and possible ways forward are indicated.

9.3.1. Extensions to the Framework

The framework provided by the scheme provides guidelines for the detailed consideration of the relevant physical laws, the construction of the plant, timing issues, required system behaviour and safety critical issues. However, there are several important issues which have not been considered in detail, these issues are briefly discussed below with possible extensions to the framework for their inclusion.

Fault analysis. This could be incorporated into the framework by extending the analysis methodologies at the controller level to cover the possibilities of faults occurring in the safety or mission controller. In fact, it is essential to consider the effects of faulty behaviour in safety-critical systems. The necessity to deal with faulty behaviour makes it essential to have well structured systems, so that the influence of faults can be localized. Faults are of no consequence, for the safety of the system, unless they lead to the system exhibiting hazardous behaviour. The structure imposed by partitioning the controller into a safety and a mission controller, allows the (safety) fault analysis of a system to concentrate on the safety controller (provided faults in the mission controller cannot influence the properties captured by the SED of that system). A fault analysis of the safety controller should consider faults in the actuators and sensors which could adversely influence real world behaviour and faults in the control system which could impact the behaviour of the controller variables.

Methodologies for fault analysis should also incorporate traditional safety techniques, such as HAZOPS, FTA and FMECA.

Probabilistic analysis. This could be incorporated into the framework by introducing a probabilistic dimension into the formal model. A possible approach for the inclusion of probabilities into the formal model would be to associate probabilities with the relations that reflect the degree of belief that the relation is satisfied by an arbitrary system history.

The methodologies should then be extended to exploit the extra information provided by the introduction of the probabilities, for example, to make comparisons between different controller specifications.

Employment of redundancy. The separation of the safety and mission issues suggests that redundancy is best employed in the safety controller. Methodologies could be added to determine whether for a given system redundancy should be employed at the component level or as redundant safety controllers. The knowledge obtained from fault analysis and probabilistic analysis should be exploited by such methodologies.

Broaden system class. A detailed consideration of systems for which the safety controller monitors the behaviour of the mission controller, to prevent the mission controller from performing hazardous actions, should be performed. For such systems, it is up to the safety controller to monitor the operating conditions of the mission controller and then to issue appropriate control commands. Hence the influence of the mission controller is under the control of the safety controller. Furthermore, the properties of the “sensors” used by the safety controller to monitor the mission controller must be captured by the safety environment description.

The framework could be extended to consider systems in which a clear distinction between the safety and mission issues cannot be made. In particular, it was assumed that separate sensors and actuators could always be provided for the safety controller and mission controller. The approach could be extended to cover systems for which this assumption does not hold. One possible approach would be to introduce “shared” actuators and sensors, that is actuators and sensors that are used by both controllers, but where the safety controller restricts access of the mission controller to the “shared” actuators.

The framework should be extended to allow a detailed study of distributed systems. A possible approach is to classify distributed systems into two classes, according to whether communication between the different nodes of the system is required for safety. The first class being those for which communication is required between the safety controllers of

the nodes and the second those systems for which communication is not required. For the second class the investigation could proceed by identifying protocols for the communication between the safety controllers.

Subsequent development. The analysis could be extended to consider the control systems of the controllers. That is, the development of the control system specifications. If possible, verification conditions should be formulated from which production rules can be derived to guide the analysis and production of the specifications at the control system level.

Automated support. For the framework to be useful in practical systems some automated tools that support the methodologies of the framework should be available. A first generation of support tools should provide means to concisely handle the information produced by the analysis. A second generation of tools should provide automated support for the confirmation of the verification, completeness and consistency checks.

9.3.2. Investigation of Other Formalisms

It would be interesting to investigate how the development methodology could be used to enhance the role of other formalisms during requirements analysis. It is believed that the basic principles of the framework would be applicable to different formalisms. In particular, the framework may support the use of more than one formalism during requirements analysis.

Some preliminary work in this area suggests that logical formalisms (some form of real-time temporal logic) would be most suitable for real world specifications, since the behaviour could be expressed in terms of all possible runs (c.f., semantics of the satisfaction conditions). However, the lack of a structuring concept (such as that of a mode) may cause problems during the production of the mission real world specification. It is also indicated that graphical formalisms (such as a form of timed Petri-nets) are more suitable for controller specification, since they can better represent the interactions between the components of the controllers. However, the lack of a structure in the mission real world specification (expressed using temporal logic) may cause difficulties in the verification of the controller analysis.

To define verification conditions for the controller specification and derive construction rules for the controller analysis, a scheme to connect the two formalisms would be required. From a theoretical perspective, it would be interesting to discover if two different formalisms can be incorporated into an integrated formal method in such a way that the most appropriate features of each formalism are exploited at each stage of the analysis.

The primary aim of the research, presented in this thesis, was to investigate the role of formal methods in requirements analysis. This thesis has shown that formal methods can play a useful role in requirements analysis. But to play a useful role a formal method must provide a framework for requirements analysis as a related system development model (i.e. what specifications should be produced), development methodology (i.e. how the specifications should be produced) and a formal model (i.e. a means to precisely state the specifications and allow formal analysis).

References

- [Abde86] A. A. Abdel-Ghaly, P. Y. Chan and B. Littlewood, "Evaluation of Competing Software Reliability Predictions", *IEEE Transactions on Software Engineering*, Vol. 12, No. 9, pp. 950–967. September 1986.
- [ACAR86] ACARD, "Software – A Vital Key to UK Competitiveness", an ACARD report, 1986.
- [Ande79] T. Anderson and B. Randell, "Computing Systems Reliability", Cambridge University Press, England, 1979.
- [Ande81] T. Anderson and P. A. Lee, "Fault Tolerance: Principles and Practice", Prentice–Hall Englewood Cliffs, New Jersey, 1981.
- [Ande85] T. Anderson, "Software Fault Tolerance, an Evaluation" *IEEE transactions on Software Engineering*, Vol. 11, No. 12, pp. 1491–1500, December 1985.
- [Arla88] J. Arlat, K. Kanoun and J. C. Laprie, "Dependability Evaluation of Software Fault–Tolerance", *Proceedings of FTCS–18*, pp. 142–147, Tokyo, Japan, June 1988.
- [Barn89] M. Barnes, "Dependable Computing in the UK". *International Working Conference on Dependable Computing For Critical Applications*, pp. 7–14, Santa Barbara, CA, August 1989.
- [Benh80] J. Van Benthem, "Points and Periods", in: *Time, Tense and Quantifiers, Proceedings of the Stuttgart conference on the Logic of Tense and Quantification*, pp. 40–57 Max Niemeyer Verlag, 1980.
- [Bern81] A. Bernstein and P. K. Harter, "Proving Real–Time Properties of Programs with Temporal Logic", *Proceedings of the 8th symposium on operating principles*, pp. 1–11, December 1981.
- [Bish86] P. Bishop, "Invariants as an alternative to Petri Nets for safety design", EWICS TC7, WP 498, 1986.
- [Bish86] P. G. Bishop et al. "PODS – A Project on Diverse Software" *IEEE transactions on Software Engineering*, Vol. 12, No. 9, pp. 929–940, September 1986.

References

- [Boeh75] B.W. Boehm, R.L. McClean, and D.B. Urfig, "Some Experiences with Automated Aids to the Design of Large-Scale Reliable Software", *IEEE transactions on Software Engineering*, Vol. 1, No. 2, pp. 929-940, February 1975.
- [Bugr84] J.P. Burgess, "Basic tense logic", in: *Handbook of Philosophical Logic*, eds. D. Gabbay and F. Guenther, D.Reidel, pp. 89-113, 1984.
- [Camp79] R. H. Campbell, K. H. Horton and G. G Belford, "Simulations of a Fault Tolerant Deadline Mechanism", *Proceedings of FTCS-9*, Madison, USA, pp. 95-101, June 1979.
- [Cha86] S. D. Cha, "A Recovery Block Model and its analysis", *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1986*, pp. 21-26, Sarlat, France, October 1986.
- [Chan85] M. Chandrasekharan, B. Dasarthy and Z. Kishimoko, "Requirements based testing of real-time systems: modelling for testability", *IEEE computer*, Vol. 18, No. 4, pp. 71-80, April 1989.
- [Chem77] Chemical Industry Safety and Health Council, "*A Guide to Hazard and Operability Studies*", Chemical Industry Safety and Health Council of the Chemical Industries association Limited, 1977.
- [Cour89] M. Courvoisier, R. Valette, A. Sahraoui and M. Combacau, "Specification and Implementation Techniques for Multilevel Control and Monitoring of F.M.S", in: *Computer Applications in Production and Engineering*, eds. F. Kimura and A. Rolstadas, Elsevier Science Publishers B. V. (North-Holland) IFIP, pp. 509-516, 1989.
- [Crow77] L. H. Crow, *Confidence Interval Procedures for Reliability Growth Analysis*, Tech. Report No. 197, US Army Material Systems Analysis Activity, Aberdeen, Md, 1977.
- [Cris90] F. Cristian, "Fault Tolerance in the Advanced Automation System", *Proceedings of FTCS-20*, Newcastle upon Tyne, UK, pp. 6-17, June 1990.
- [Cull88] W. J. Cullyer, "High-Integrity Computing", in: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 331, ed. M. Joseph, Springer-Verlag, pp. 1- 35, 1988.

References

- [Dasa85] B. Dasarathy, "Timing Constraints of Real-Time Systems: Constructs for Expressing them, Methods of Validating them", *IEEE Transactions on Software Engineering*, Vol SE-11, No. 1, pp. 80-86, January 1985.
- [Davi79] A. M. Davis, T. G. Rauscher, "Formal techniques and automatic processing to ensure correctness in requirements specifications", *Proceedings of the conference on specifications of reliable software*, IEEE computer society, pp. 15-35, 1979.
- [Dunh81] J.R. Dunham and J.C. Knight (editors), "Production of reliable flight-crucial software", *Proceedings of Validation Methods Research for Fault-Tolerant Avionics and Control Systems Sub-Working-Group Meeting*, Research Triangle Park, North Carolina, November, 2-4, 1981, NASA Conference Publication 2222.
- [ECFE85] ECFE, "Risk analysis in the process industries", *The Institution of Chemical Engineers*, 1985.
- [End75] A.B. Endres, "An analysis of errors and their causes in software systems", *IEEE transactions on Software Engineering*, Vol. 1, No. 2, February 1975.
- [Emer86] E. A. Emerson and J. Y. Halpern. " 'Sometimes' and 'not never' revisited: on branching versus linear time temporal logic. *Journal of the Association for Computing Machinery*, Vol. 33, No. 1, pp. 151-178, January 1986.
- [Equ90] Equinox, "Fly by Wire II", Channel 4, September 1990.
- [Eric81] C. A. Ericson, "Software and system safety", *Proceedings of the fifth International System Safety Conference*, Vol. 1, Part 1, pp. III-B-1 to III-B-11 Denver, 1981.
- [EWIC85] EWICS TC7, "System Requirements Specification for Safety Related Systems", January 1985.
- [Fore90] T. Forester and P. Morrison, "Computer Unreliability and Social Vulnerability", *Futures*, pp. 462-474, June 1990.
- [Gers87] L. J. Gerstein, *Discrete Mathematics and Algebraic Structures*, W. H. Freeman, 1987.

References

- [Goel85] A. L. Goel, "Software Reliability Models: Assumptions, and Applicability", *IEEE transactions on Software Engineering*, Vol. 11, No. 12, pp. 1411–1423, December 1985.
- [Gors86] J. Gorski, "Design for safety using temporal logic", *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1986*, Sarlat, France, pp. 149–155, October 1986.
- [Gors88] J. Gorski, "Formal specification of real time systems", *Computer Physics Communications*, Vol. 50. No. 1–2, pp. 71–88, 1988.
- [Grig81] J. G. Griggs, "A method of software safety analysis", *Proceedings of the fifth International System Safety Conference*, Denver, 1981, Vol. 1, Part 1, pp. III–D–1 to III–D–18.
- [Hare87] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", *Science of Programming* 8, 1987.
- [Hat187] D. J. Hatley and I. A. Pirbhai, "*Strategies for Real-Time System Specification*", Dorset House Publishing, New York, 1987.
- [Heni80] K.L. Heninger, "Specifying Software Requirements for Complex Systems: New techniques and their applications", *IEEE transactions on Software Engineering*, Vol. 6, No. 1, pp. 2–13, January 1980.
- [HSE87] HSE. "*Guidelines on Programmable Electronic Control Systems in Safety-Related Applications, PARTS 1 & 2*". Her Majesty's Stationery Office, London, 1987.
- [Ho89] Y. Ho, "Dynamics of Discrete Event Systems", *Proceedings of the IEEE*, pp. 3–6, January 1989.
- [Hope83] S. Hope, "Methodologies for hazard analysis and risk assessment in the petroleum refining and storage Industry" *Hazard prevention* (Journal of the System Safety Society), pp. 24–32, July/August 1983.

References

- [Jaff89] M.S. Jaffe. and N.G. Leveson, "Completeness, Robustness, and Safety in Real-Time Software Requirements Specification", *Proc 11th International Conference on Software Engineering*, Pittsburgh, PA, pp. 302–311, May 1989.
- [Jaha86] F. Jahanian and A.K. Mok, "Safety Analysis of Timing Properties in Real-Time systems", *IEEE transactions on Software Engineering*, Vol. 12, No. 9, pp. 890–904, September 1986.
- [Jaha88] F. Jahanian and D. A. Stuart, "A Method for Verifying Properties of Modechart Specifications", *Proceedings of IEEE Real-Time Systems Symposium*, pp. 12–21, December 1988.
- [Jara88] J. Jaray, "Timed Specifications for the development of real-time systems", in: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 331, ed. M. Joseph Springer-Verlag, pp. 67– 83, 1988.
- [Jeli72] Z. Jelinski, Z. and P .B. Moranda, "Software Reliability Research", in: *Statistical Computer performance Evaluation*, ed. W. Frieberger, Academic Press, New York, pp. 465–484, 1972.
- [Jose89] M. Joseph and A. Goswami, "Relating Computation and Time", *Proceedings of the Joint University of Newcastle upon Tyne and ICL seminar on Real-Time Systems*, University of Newcastle upon Tyne Computing Laboratory pp. III.1– III.15, September 1989.
- [Kell86] A. Keller, "Safety and Reliability engineering advances", *Control and Instrumentation*, vol. 18, pp. 135–136, June 1986.
- [Keil83] P. A. Keiller, B. Littlewood, D. R. Miller, A. Sofer, "Comparison of Software Reliability Predictions", *Proceedings of FTCS-13*, Milan, Italy, pp. 128–134, 1983.
- [Knig86] J.C. Knight and N.G. Leveson, "An Experimental Evaluation of Software Fault-Tolerance", *IEEE transactions on Software Engineering*, Vol. 12, No. 1, pp. 96–109, January 1986
- [Kuhn89] S. T. Kuhn, "Tense and Time", in: *Handbook of Philosophical Logic (iv)*, D. Gabbay and F. Guenther (Eds), pp. 513–532, 1989.

References

- [Lapr85] J. C. Laprie, "Dependable computing and fault tolerance: concepts and terminology", *Proceedings of FTCS-15*, Ann Arbor, Michigan, pp. 2-11 June 1985.
- [Leve82] N.G. Leveson and P.R. Harvey, "Software safety", *ACM Software Engineering notes*, Vol. 7, No. 2, pp. 21-26, April 1982.
- [Leve83a] N.G. Leveson and T. Shimeall, "Safety assertions for process control systems", *Proceedings FTCS-13*, Milan, Italy, 1983.
- [Leve83b] N.G. Leveson and P.R. Harvey, "Analyzing Software safety", *IEEE transactions on Software Engineering*, Vol. 9, No. 5, pp. 569-579, September 1983.
- [Leve84] N.G. Leveson, "Software safety in computer-controlled systems", *IEEE Computer*, Vol. 17, No. 2, pp. 48-55, February 1984.
- [Leve85] N. Leveson, "Software Safety", in: *Resilient Computing Systems*, ed. T. Anderson, Collins, pp. 122-143, 1985.
- [Leve86] N.G. Leveson, "Software Safety: Why, what and how", *ACM Computing Surveys*, Vol. 18, No. 2, pp. 125-163, June 1986
- [Leve87] N.G. Leveson and J.L. Stolzy. "Safety Analysis Using Petri Nets". *IEEE Transactions on Software Engineering*, Vol. 13, No. 3, March 1987.
- [Leve89] N.G. Leveson, "Safety-Critical Software Development", in: *Safe and Secure Computing Systems*, ed. T. Anderson. Blackwell Scientific Publications, pp. 155-162. 1989
- [Litt73] B. Littlewood and J. L. Verral, "A Bayesian Reliability Growth Model for Computer Software", *J. Royal Statist. Soc.*, pp. 332-346, 1973.
- [Litt81] B. Littlewood, "Stochastic Reliability Growth: a model for fault removal in Computer Programs and Hardware Designs", *IEEE Transactions on Reliability*, Vol. 30, No. 4, April 1981.
- [Litt85] B. Littlewood, "Software reliability Prediction", in: *Resilient Computing Systems*, ed. T. Anderson, Blackwell Scientific Publications, pp. 144-162, 1985.

References

- [Long77] A. B. Long, "A methodology for the development and validation of critical software for nuclear power plants", *Proceedings of Compsac 1977*, Chicago, Illinois, pp. 620–627, November 1977.
- [McDe90] J. A. McDermid, "Skills and Technologies for the Development and Evaluation of Safety Critical Systems", *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1990*, Gatwick, London, November 1990 (to be published).
- [MacE88] G. H. MacEwen, David B. Skillcorn, "Using Higher-Order Logic for Modular Specification of Real-Time Distributed Systems", in: *Formal Techniques in Real-Time and Fault-Tolerant Systems. Springer-Verlag, LNCS 331*, ed. M. Joseph pp. 36– 66, 1988.
- [Maib87] T. S. E. Maibaum, "A Logic for the Requirements Specification of Real-Time Embedded Systems", FOREST project report, Imperial College of Science and Technology, London, UK, 1987.
- [Merl76] P. M. Merlin and D. J. Farber, "Recoverability of Communication Protocols—Implications of a Theoretical Study", *IEEE Transactions on Communications*, pp. 1036–1043, September 1976.
- [Miln83] R. Milner, "Calculi for Synchrony and Asynchrony", *Theoretical Computer Science*, Vol. 25, pp 267–310, 1983.
- [Mok85] A. K. Mok, "SARTOR – A design environment for real-time ", *Proceedings of Compsac 85*, Chicago, Illinois, pp. 174–182, October 1985.
- [Morg89] A. Morgan and R. Matthews, "French reactor threat to UK", *The Sunday Correspondent*, pp 3–3, December 3 1989,
- [Mose90] Moser and Melliard-Smith, "Formal Verification of Safety-critical Systems", *Software Practice and Experience*, Vol. 20, No. 8, pp. 799–821, August 1990.
- [Mull79] G.P. Mullery, "CORE – A Method for Controlled Requirement Specification", *Proc. 4th International Conference on Software Engineering*, Munich, Germany, pp. 126–135, September 1979.

References

- [Mulz85] M. Mulzanni, "Reliability versus Safety", *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1985*, Como, Italy, pp 149–155, October 1985.
- [Nagl83] J. Nagle and S. Jhonson, "Automatic program proving for real-time embedded software", *Proceedings of the 10th annual symposium on principles of programming languages*, pp. 48–56, 1983.
- [Nunn86] S. R. Nunns, D. A. Mills and G. C. Tuff, "Programmable Electronic Systems Safety: Standards and Principles – An Industrial Viewpoint", *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1986*, Sarlat, France, pp. 17–20, October 1986.
- [Ostr87] J. S. Ostroff and W. M. Wonham, "Modelling, specifying and verifying real-time embedded computer systems", *Proceedings of the 8th IEEE Real-Time System Symposium*, San Jose, CA, pp. 124–132, December 1987.
- [Ostr89a] J. S. Ostroff, *"Temporal Logic for Real-Time Systems"*. Advanced Software Development Series, Research Studies Press Limited. England Wiley, 1989.
- [Ostr89b] J. S. Ostroff, *"Synthesis of controllers for real-time discrete event systems"*, York University, Ontario, Canada, Technical Report No. CS-89-09, June 1989.
- [Ostr90] J. S. Ostroff, "Deciding Properties of Timed Transition Models", *IEEE transactions on parallel and distributed systems*, Vol. 1. No. 2, pp. 170–183, April 1990.
- [Parn88] D.L. Parnas, A.J. van Schouwen, Shu Po Kwan, "Evaluation Standards for Safety Critical Software", *Technical Report 88-220*, Queens University, Kingston Ontario, Canada, May 1988.
- [Perr84] C. Perrow, *Normal Accidents: Living with High Risk Technologies*, New York: Basic Books, 1984.
- [Pnue86] A. Pnueuli, "Specification and Development of Reactive Systems", *IFIP 86*, pp. 845– 857, 1986.

References

- [Pnue88] A. Pnueli and E. Harel, "Applications of Temporal Logic to the Specification of Real Time Systems", in: *Formal Techniques in Real-Time and Fault-Tolerant Systems*. Springer-Verlag, LNCS 331, ed. M. Joseph, pp. 84– 97, 1988.
- [Pott86] C. Potts, A. Finkelstein, M. Aslett, J. Booth, "*Structured Common Sense: A Requirements Elicitation and Formalization Method for Modal Action Logic*", FOREST project report, Imperial College of Science and Technology, London, UK, 1986.
- [Pucc90] G. Pucci, "On the Modelling and Testing of Recovery Block Structures", *Proceedings of FTCS-20*, Newcastle upon Tyne, UK, pp. 356–363, June 1990.
- [Quir85] W.J. Quirk. "*Verification and validation of real-time software*", Springer Verlag, 1985.
- [Quir86] W. J. Quirk, "Engineering Software Safety", *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1986*, Sarlat, France, pp. 143–147, October 1986.
- [Ramc74] C. Ramchandani, "*Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*", TR 120, Project MAC, MIT, February 1974.
- [Reas87] J. Reason, "The Chernobly errors", *Bulletin of the British Psychological Society*, April 1987.
- [Reed86] G. M. Reed and A.W. Roscoe, "A timed model for communicating sequential processes", LNCS 226, Springer-Verlag, 1986.
- [Reev86] P. Reeves, *Control and Instrumentation*, pp. 49, Feb. 1986.
- [Redm85] F. Redmill, "*Proposals for designing for safety*", EWICS TC7, WP 438, 1985.
- [Ridl83] J. Ridley, *Safety at Work*, Butterworths, London, 1983.
- [Rodg71] W. P. Rodgers, "*Introduction to System Safety Engineering*", New York, Wiley, 1971.
- [Rola83] H. E. Roland and B. Moriarty, "*System Safety Engineering and Management*", John Wiley & Sons, 1983.
- [Rom85] G.C. Roman, "A Taxonomy of Current Issues in Requirements Engineering", *IEEE computer*, Vol. 18, No. 4, pp 14–23, April 1985.

References

- [Rouq86] J. C. Rouquet and P. J. Traverse, "Safe and Reliable Computing on Board the Airbus and ATR aircraft", *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1986*, Sarlat, France, pp. 17–20, October 1986.
- [Rous81] W. B. Rouse, "Human–computer interaction in the control of dynamic systems", *ACM computing surveys*, Vol. 13, No. 1, pp. 71–100, March 1981.
- [Roys77] H. S. Royston, "The content and Implications of the ACT", *The Health and Safety at Work act and its effect on Industry*, pp 1–8, 1977.
- [Rung86] B. Runge, "Quantitative Assessment of Safe and Reliable Software", *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1986*, Sarlat, France, pp. 7–11. October 1986.
- [Rzep85] W. Rzepka and Y. Ohuno, "Requirements Engineering: Software Tools for Modelling User Needs", *IEEE computer*, Vol. 18, No. 4, pp. 9–12, April 1989.
- [Safe79] *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1979*.
- [Sae90] A. Saeed, T. Anderson and M. Koutny, "A Formal Model for Safety–Critical Computing Systems", *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1990*, Gatwick, London, November 1990 (to be published).
- [Sag186] F. Sagletti and W. Ehrenberger, "Software Diversity – Some Considerations about its Benefits and Limitations", *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1986*, Sarlat, France, pp. 27–34, October 1986.
- [Scot87] R.K. Scott, J. W. Gault and D. F. McAllister, "Fault–Tolerant Software Reliability Modelling", *IEEE Transactions on Software Engineering*, Vol. 13, No. 5, pp. 96–109, May 1987.
- [Scot83] R.K. Scott, J. W. Gault and D.F. Mc Allister, "The Consensus Recovery Block", *Proceedings Total Systems Reliability Symposium*, Gaithersburg, pp. 74–85, December 1983.
- [Smit72] C. L. Smith, "*Digital Process Control*", Intext Educational Publishers, London, 1972.

References

- [Stru89] N. Strutt, “*A Survey Of Formal Methods*”, Report from the Ministry of Defence, ARE TM (AXC) 89001, January 1989.
- [Somm82] I. Sommerville, “*Software Engineering*”, Addison–Wesley Publishers, London, 1982.
- [Tayl89] J. R. Taylor, “Very High Reliability Computer Systems”, in: *Safe and Secure Computing Systems*, ed. T. Anderson, Blackwell Scientific Publications, pp. 244–248, 1989.
- [Theu86a] N. Theuretzbacher, “Using AI–Methods to Improve Software Safety”, *Proceedings of the IFAC Workshop on Safety of Computer Control Systems 1986*, Sarlat, France, pp. 99–105, October 1986.
- [Theu86] N. Theuretzbacher, “VOTRICS: Voting Triple Modular Computing System”, *Proceedings of FTCS–16*, Vienna, Austria, June 1986.
- [Uram77] A. B. Long, “Computer control in a combined cycle power plant”, *Proceedings of Compsac 1977*, Chicago, Illinois, pp. 608–614, November 1977.
- [Vend80] V. F. Venda and B. F. Lomov. “Human factors leading to engineering safety systems”, *Hazard Prevention* (Journal of the System Safety Society), pp. 6–13, March/April 1980.
- [Vese81] W. E. Vesely, F. F. Goldberg, N. H. Roberts and D. F. Haasl, “*Fault Tree Handbook*”, NUREG–0492, U.S. Nuclear Regulatory Commission, January 1981.
- [Wass79] A. Wasserman, S. K. Stinson, “A specification method for Interactive information systems”, *Proceedings of the conference on specifications of reliable software*, IEEE computer society, pp. 68–79, 1979.
- [Wing90] J. M. Wing, “A specifiers introduction to formal methods”, *IEEE Computer*, Vol. 23, No. 9, pp. 8–21, September 1990.
- [Wupp88] H. Wupper and J. Vytopil, “A Specification Language for Reliable Real–Time Systems”, in: *Formal Techniques in Real–Time and Fault–Tolerant Systems, LNCS 331*, Springer–Verlag, ed. M. Joseph. pp. 84– 97, 1988.

References

- [Yeh80] R.T. Yeh and R. T. Mittermeir, "Conceptual Modeling as a Basis for Deriving Software Requirements", *International Computer Symposium*, Taipei, Taiwan, Dec. 1980.
- [Youn82] R. E. Young, "Control in hazardous environments", *IEE Control Engineering Series*, no. 17, 1982.

Appendix A - Glossary of Symbols and Definitions

Time

BT	<i>Base Time.</i> The time set for the specification model (the set of non-negative reals).
t, t_0, t_1, \dots	<i>Time Points.</i> $t \in BT$.
Int, Int_0, Int_1, \dots	<i>Time Intervals.</i> $Int \subseteq BT$ is a <i>time interval</i> iff $\forall t_0, t_1 \in Int: \forall t_2 \in BT: (t_0 < t_2 < t_1 \Rightarrow t_2 \in Int)$.
$SI(Int)$.	<i>Interval Set.</i> The set of all intervals within the interval Int .
$s(Int)$	<i>Start Point.</i> $s(Int) = t$ s.t. $\forall t_0 \in Int: t \leq t_0 \wedge (\forall t_1: (\forall t_0 \in Int: t_1 \leq t_0) \Rightarrow t_1 \leq t)$.
$e(Int)$	<i>End Point.</i> $e(Int) = t$ s.t. $\forall t_0 \in Int: t \geq t_0 \wedge (\forall t_1: (\forall t_0 \in Int: t \geq t_0) \Rightarrow t_1 \geq t)$.
$dur(Int)$	<i>Duration.</i> $dur(Int) = e(Int) - s(Int)$.
$T(SY)$	<i>System Lifetime.</i> The interval of time that represents the lifetime of system SY .

System State

p_1, p_2, \dots	<i>State Variables.</i> The variables that define the state of the system.
$SY.Sv$	<i>State Vector.</i> The vector of all the state variables of system SY .
$SY.n$	<i>State Vector Number.</i> $n = Sv(SY) $.
$SY.S$	<i>State Set.</i> The set of all the state variables of system SY .
V_{p_i}	<i>Variable Range.</i> The set of possible values for state variable p_i .
$SY.\Gamma$	<i>State Space.</i> The set of possible values for state vector of system SY .
V, V_1, V_2	<i>State Values.</i> Values from the state space of a system.

SY.P, SY.Pv	<i>Physical Variables.</i> The set (resp. vector) of variables of system SY that represent the state of the physical process.
SY.Op, SY.Opv	<i>Operator Variables.</i> The set (resp. vector) of variables of system SY that represent the state of the operator console.
SY.Td, SY.Tdv	<i>Transducer Variables.</i> The set (resp. vector) of variables of a system SY that represent the state of the sensors and actuators.
SY.R SY.Rv	<i>Real world Variables.</i> The set (resp. vector) of physical and operator variables of system SY.
SY.SR, SY.SRv	<i>Safety Real world Variables.</i> The set (resp. vector) of real world variables which affect the critical behaviour of system SY.
SY.MR, SY.MRv	<i>Mission Real world Variables.</i> The set (resp. vector) of real world variables which affect the mission oriented behaviour of system SY.
SY.C SY.Cv	<i>Controller Variables.</i> The set (resp. vector) of transducer and operator variables of system SY.
SY.SC, SY.SCV	<i>Safety Controller Variables.</i> The set (resp. vector) of controller variables which affect the critical behaviour of system SY.
SY.MC, SY.MCv	<i>Mission Controller Variables.</i> The set (resp. vector) of controller variables which affect the mission oriented behaviour of system SY.

Remark: If the system is obvious from the context, or the discussion is for any system SY is omitted from the notation.

System Behaviour

H, H _x , H _y , ...	<i>Histories.</i> Functions from the lifetime of a system to its state space.
ΓH	<i>Universal History Set.</i> The set of all functions of the form $H: T \rightarrow \Gamma$, for a given system.
HH	<i>History Set.</i> $HH \subseteq \Gamma H$.

$H _{Int}$	<i>Restricted Domain History.</i> The domain of H is restricted to Int .
$H.pv$	<i>Restricted Range History.</i> The range of H is restricted the possible values of vector pv .
Cp_i	<i>Variable Class Relation.</i> A predicate over a free function variable p_i .
$H \text{ sat } Cp_i$	<i>Class satisfaction.</i> The fact that a history H satisfies the class relation Cp_i .
$SysPred$	<i>System Predicate.</i> A predicate over n free value variables p_1, p_2, \dots, p_n of type V_{p_1}, \dots, V_{p_n} .
$H \text{ sat } CP$	<i>Variable class satisfaction</i> $H \text{ sat } Cp \text{ iff } H \text{ sat } Cp_i, \text{ for all } i \in \{1, \dots, n\}.$
$V \text{ sat } SysPred$	<i>System Predicate Satisfaction.</i> The fact that a system predicate is satisfied for a state V .
Ir_1, Ir_2, \dots	<i>Invariant relation.</i> A system predicate Ir is an invariant relation for a history H iff Ir is satisfied for all states given by H . This is denoted by $H \text{ sat } Ir$.
$HistPred$	<i>History Predicate.</i> A predicate over two free time variables $T_0, T_1, 2..n$ free value variables $p_{1,0}, \dots, p_{n,0}, p_{1,1}, \dots, p_{n,1}$ (where $p_{i,j}$ has type V_{p_i}), n free function variables p_1, \dots, p_n (where p_i is a function of class Cp_i).
$H \text{ sat } HistPred@Int$	<i>History Predicate Satisfaction.</i> The fact that a history H satisfies a history predicate $HistPred$ for an interval Int .
$Hr_1, Hr_2, ..$	<i>History Relation.</i> A history predicate Hr is a history relation for a history H iff $\forall Int \in SI(T): Hr \text{ sat}@Int$. This is denoted by $H \text{ sat } Hr$.
$Desc$	<i>History Description.</i> A six-tuple $Desc = \langle T, Sv, VP, CP, IR, HR \rangle$, where T is the system lifetime; Sv is the state vector; VP is the sequence of variable ranges; CP is the sequence of variable class relations, $\langle Cp_1, \dots, Cp_n \rangle$; IR is the

sequence of invariant relations, $\langle Ir_1, \dots, Ir_m \rangle$ and HR is the sequence of history relations, $\langle Hr_1, \dots, Hr_k \rangle$

D, D_1, D_2, \dots *History Descriptions.*

$Set(Desc)$ *History Description Set.*

$Set(Desc) = \{H \in \Gamma H \mid H \text{ sat } CP \wedge H \text{ sat } C(IR) \wedge H \text{ sat } C(HR)\},$
where $C(IR) = Ir_1 \wedge \dots \wedge Ir_m$, $C(HR) = Hr_1 \wedge \dots \wedge Hr_k$.

$Iset(Desc)$ *Invariant Description Set.*

$Iset(Desc) = \{H \in \Gamma H \mid H \text{ sat } C(IR)\}.$

Perfect clock *Perfect Clock Class.*

A variable $p_i \in P$ is a perfect clock for a description D iff :
 $Cp_i(D) = (\forall t \in T: p_i = t).$

$H \text{ sat SysPred}@t$ *Point Satisfaction.*

$H \text{ sat SysPred}@t$ iff $\langle H.p_1(t), \dots, H.p_n(t) \rangle \text{ sat SysPred}.$

$H \text{ sat SysPred}@Int$ *Interval Satisfaction.*

$.H \text{ sat SysPred}@Int$ iff $H \text{ sat SysPred}@t$ for all $t \in Int$.

$H \text{ sat SysPred}\odot Int$ *Event Satisfaction.*

$H \text{ sat SysPred}\odot Int$ iff

$H \text{ sat } \neg \text{SysPred}@t$ for all $t \in Int - \{e(Int)\} \wedge H \text{ sat SysPred}@e(Int).$

Time Bound Definition.

Constraint

A time bound constraint Tb is any predicate built using standard logical connectives; standard mathematical functions; and a free variable Iv of type $SI(T)$. No other free variables can be used.

$Int \text{ sat } Tb$

Time bound constraint satisfaction.

.We will say that an interval Int satisfies a time bound constraint Tb if and only if the substitution of Int for Iv into Tb leads to a well-defined Boolean expression which evaluates to true.

Ω *Termination Predicate.* $\Omega = (p_i = e(T))$, where p_i is a perfect clock.

Mode Theory

Mode	<p><i>Definition.</i></p> <p>A five-tuple, $\text{Mode} = \langle \text{Start}, \text{Inv}, \text{End}, \text{LB}, \text{UB} \rangle$, where Start, Inv and End are system predicates, and LB and UB are time values (or time valued functions).</p>
m, m_1, m_2, \dots	<i>Modes.</i>
$\text{Start}(m)$	<i>Start Predicate.</i> The predicate Start of the mode specification of m .
$\text{Inv}(m)$	<i>Invariant Predicate.</i> The predicate Inv of the mode specification of m .
$\text{End}(m)$	<i>End Predicate.</i> The predicate End of the mode specification of m .
$\text{LB}(m)$	<i>Lower Bound.</i> The time value LB of the mode specification of m .
$\text{UB}(m)$	<i>Upper Bound.</i> The time value UB of the mode specification of m .
$H \text{ sat } m@Int$	<p><i>Mode Satisfaction.</i></p> <p>$H \text{ sat } m@Int$ iff</p> <p>$H \text{ sat } \text{Start}(m)@s(Int) \wedge H \text{ sat } \text{Inv}(m)@Int \wedge H \text{ sat } \text{End}(m)\odot Int$</p> <p>$\wedge Int \text{ sat } (\text{dur}(Iv) \geq \text{LB} \wedge \text{dur}(Iv) \leq \text{UB})$.</p>
$H \text{ sat } \mathcal{J}(m)@Int$	<p><i>Start Satisfaction.</i></p> <p>$H \text{ sat } \mathcal{J}(m)@Int$ iff</p> <p>$H \text{ sat } \text{Start}(m)@s(Int) \wedge H \text{ sat } \text{Inv}(m)@Int \wedge$</p> <p>$(H \text{ sat } \neg \text{End}(m)@Int \vee H \text{ sat } \text{End}(m)\odot Int)$.</p>
$H \text{ sat } \mathcal{E}(m)@Int$	<p><i>End Satisfaction.</i></p> <p>$H \text{ sat } \mathcal{E}(m)@Int$ iff $H \text{ sat } \text{Inv}(m)@Int \wedge H \text{ sat } \text{End}(m)\odot Int$</p>
$H \text{ res } m@Int$	<i>Mode Residence.</i> $H \text{ res } m@Int_0$ iff $\exists Int_1: H \text{ sat } m@Int_1 \wedge Int_0 \subseteq Int_1$.
$m_1 \simeq^{HH} m_2$	<p><i>Mode Equivalence.</i></p> <p>$m_1 \simeq^{HH} m_2$ iff</p> <p>$\forall H \in HH: \forall Int \in SI(T): [H \text{ sat } m_1@Int \Leftrightarrow H \text{ sat } m_2@Int]$.</p>

$m_1 \approx^{HH} m_2$	<p><i>Mode Implication.</i> $m_1 \approx^{HH} m_2$ iff</p> <p>$\forall H \in HH: \forall Int \in SI(T): [H \text{ sat } m_1@Int \Rightarrow H \text{ sat } m_2@Int].$</p>
$m \text{ con } D$	<p><i>Mode Consistency.</i></p> <p>$m \text{ con } D$ iff $\exists H \in Iset(D): \exists Int \in SI(T): H \text{ sat } m@Int.$</p>
Mod	<i>Modes Set.</i> A set of modes.
ModeSeq	<p><i>Definition.</i></p> <p>A finite sequence of modes, $ModeSeq = \langle m_1, ..., m_r \rangle$, where m_i is a mode, for $i \in \{1, ..., r\}.$</p>
$ms, ms_1, ..$	<i>Mode Sequences.</i>
$ ms $	<i>Mode sequence Length.</i> The number of modes in mode sequence ms .
$H \text{ sat } ms@Int$	<p><i>Mode Sequence Satisfaction.</i></p> <p>$H \text{ sat } ms@Int$ iff</p> <p>$\exists t_0, ... t_{ ms }: t_0 \leq ... \leq t_{ ms } \wedge t_0 = s(Int) \wedge t_{ ms } = e(Int) \wedge$</p> <p>$H \text{ sat } m_{i+1}@[t_i, t_{i+1}], \text{ for } i = 0, ..., ms - 1.$</p>
$H \text{ sat } ms$	<i>Mode Sequence Lifetime Satisfaction.</i> $H \text{ sat } ms$ iff $H \text{ sat } ms@T.$
$ms(i)$	<i>Mode sequence index function.</i> The i^{th} mode of mode sequence ms .
$ms(i, j)$	<i>Mode sequence slice.</i> $ms(i, j) = \langle ms(i), ms(i+1), ..., ms(j) \rangle.$
$S(ms)$	<i>Start mode.</i> $S(ms) = ms(1).$
$E(ms)$	<i>End mode.</i> $E(ms) = ms(ms).$
$H \text{ res } ms@Int$	<p><i>Mode Sequence Residence.</i></p> <p>$H \text{ res } ms@Int$ iff $(\exists Int_1 \in SI(T): H \text{ sat } ms@Int_1 \wedge Int_0 \subseteq Int_1).$</p>
$SeqStart(ms)$	<i>Sequence Start Predicate.</i> $SeqStart(ms) = Start(S(ms)).$
$SeqInv(ms)$	<i>Sequence Invariant Predicate.</i> $SeqInv(ms) = Inv_1 \vee ... \vee Inv_{ ms }.$
$SeqEnd(ms)$	<i>Sequence End Predicate.</i> $SeqEnd(ms) = End(S(ms)).$

$Hset(HH, ms)$	<i>Satisfaction Set.</i> $Hset(HH, ms) = \{H \in HH: H \text{ sat } ms\}.$
$ms_1 \simeq^{HH} ms_2$	<i>Mode Sequence Equivalence.</i> $ms_1 \simeq^{HH} ms_2$ iff $\forall H \in HH: \forall Int \in SI(T): [H \text{ sat } ms_1@Int \Leftrightarrow H \text{ sat } ms_2@Int].$
$ms_1 \approx^{HH} ms_2$	<i>Mode Sequence Implication.</i> $m_1 \approx^{HH} m_2$ iff $\forall H \in HH: \forall Int \in SI(T): [H \text{ sat } ms_1@Int \Rightarrow H \text{ sat } ms_2@Int].$
$ms \text{ con } D$	<i>Mode Sequence Consistency.</i> $m \text{ con } D$ iff $\exists H \in Iset(D): \exists Int: H \text{ sat } m@Int.$
Mod^+	<i>Universal Sequence Set.</i> The set of all sequences on Mod which have positive length.
$MSS, MSS_1,$	<i>Mode Sequence Set.</i> A set of mode sequences.
$H \text{ sat } MSS$	<i>Mode Sequence Set Satisfaction.</i> $H \text{ sat } MSS$ iff $\exists ms \in MSS: H \text{ sat } ms.$
$MSS_1 \simeq^{HH} MSS_2$	<i>Mode Sequence Set Equivalence.</i> $MSS_1 \simeq^{HH} MSS_2$ iff $\forall H \in HH: [H \text{ sat } MSS \Leftrightarrow H \text{ sat } MSS].$
$MSS \approx^{HH} MSS_2$	<i>Mode Sequence Set Implication.</i> $MSS_1 \approx^{HH} MSS_2$ iff $\forall H \in HH: [H \text{ sat } MSS_1 \Rightarrow H \text{ sat } MSS_2].$
MG	<i>Mode Graph</i> A four-tuple $MG = \langle M, A, S, E \rangle$, where M is a (finite) set of modes, A is a set of mode pairs (i.e, $A \subseteq M \times M$), S and E are non-empty subsets of M (i.e., $S, E \subseteq M, S \neq \emptyset$ and $E \neq \emptyset$).
MG, MG_1, \dots	<i>Mode Graphs.</i>
$M(MG)$	<i>Mode Set.</i> The set M from the specification of MG.
$A(MG)$	<i>Arc Set.</i> The set A from the specification of MG.
$S(MG)$	<i>Start Set.</i> The set S from the specification of MG.

$E(MG)$	<i>End Set.</i> The set E from the specification of MG .
$RTC(MG)$	<i>Reflexive Transitive Closure.</i> $(x, y) \in RTC(MG)$ iff $\exists w_0, \dots, w_n: x = w_0 \mathrel{A} w_1 \mathrel{A} w_2 \dots w_{n-1} \mathrel{A} w_n = y$.
$Seq(MG)$	<i>Mode Graph Sequence Set.</i> $Seq(MG) = \{ \text{vms} \in M(MG)^+ \mid$ $(ms(i), ms(i+1)) \in A(MG) \wedge ms(1) \in S(MG) \wedge ms(ms) \in E(MG),$ for $i = 1, \dots, ms -1 \}$
$H \text{ sat } MG@Int$	<i>Mode Graph Interval Satisfaction.</i> $H \text{ sat } MG@Int$ iff $\exists ms \in Seq(MG): H \text{ sat } ms@Int$.
$H \text{ sat } MG$	<i>Mode Graph Satisfaction.</i> $H \text{ sat } MG$ iff $H \text{ sat } MG@T$.
$HH(set, MG)$	<i>Satisfaction Set.</i> $Hset(HH, MG) = \{ H \in HH: H \text{ sat } MG \}$.
$GStart(MG)$	<i>Graph Start Predicate.</i> $GStart(MG) = \bigvee_{m \in S(MG)} Start(m)$.
$GInv(MG)$	<i>Graph Invariant Predicate.</i> $GInv(MG) = \bigvee_{m \in M(MG)} Inv(m)$.
$GEnd(MG)$	<i>Graph End Predicate.</i> $GEnd(MG) = \bigvee_{m \in E(MG)} End(m)$.
$MG.pr(m)$	<i>Predecessor Function.</i> $MG.pr(m) = \{ x \in M(MG) \mid \exists (x, m) \in A(MG) \}$.
$MG.sr(m)$	<i>Successor Function.</i> $MG.sr(m) = \{ x \in M(MG) \mid \exists (m, x) \in A(MG) \}$.
$MG \text{ cmp } D$	<i>Mode Graph completeness.</i> $MG \text{ cmp } D$ iff $\forall m \in M(MG): \forall H \in Set(D): \forall Int \in SI(T):$ $\exists x \in MG.sr(m): [H \text{ sat } m@Int \Rightarrow H \text{ sat } (Start(x) \wedge Inv(x))@e(Int)] \vee$ $MG.sr(m) = \emptyset$.
$MG \text{ con } D$	<i>Mode Graph Consistency</i> $MG \text{ con } D$ iff $\forall ms \in Seq(MG): ms \text{ con } D$.

$MG_1 \simeq^{HH} MG_2$ *Mode Graph Equivalence.*

$MG_1 \simeq^{HH} MG_2$ iff $\forall H \in HH: H \text{ sat } MG_1 \Leftrightarrow H \text{ sat } MG_2$.

$MG_1 \approx^{HH} MG_2$ *Mode Graph Implication.*

$MG_1 \approx^{HH} MG_2$ iff $\forall H \in HH: H \text{ sat } MG_1 \Rightarrow H \text{ sat } MG_2$.

$MG_1 \doteq(\alpha) MG_2$ *Mode Graph Isomorphism.*

A mode graph MG_1 is said to be isomorphic to a mode graph MG_2 if there is a bijection $\alpha: M(MG_1) \rightarrow M(MG_2)$ such that:

- i) (u, v) is in $A(MG_1)$ if and only if $(\alpha(u), \alpha(v))$ is in $A(MG_2)$, and
- ii) w is an element of $S(MG_1)$ if and only if $\alpha(w)$ is an element of $S(MG_2)$ and x is an element of $E(MG_1)$ if and only if $\alpha(x)$ is an element of $E(MG_2)$.

$MG_1 \equiv(\alpha) MG_2$ *Mode Graph Congruence.*

$MG_1 \equiv(\alpha) MG_2$ iff $MG_1 \doteq(\alpha) MG_2$ and $\forall m \in M(MG_1): \alpha(m) \simeq m$.

SEMG *Single Entry Exit Mode Graph.*

MG is a SEMG iff

$|S(MG)| = 1 \wedge |E(MG)| = 1 \wedge \forall m \in S(MG): MG.pr(m) = \emptyset \wedge \forall m \in E(MG): MG.sr(m) = \emptyset$.

Phase Graph *Definition*

A four-tuple, $PHG = \langle PH, A, S, E \rangle$, where PH is a (finite) set of Phases, A is a set of mode pairs (i.e., $A \subseteq PH \times PH$), S and E are non-empty subsets of PH (i.e., $S, E \subseteq PH, S \neq \emptyset$ and $E \neq \emptyset$).

Predicate Mode *Definition*

Graph A pair, $PG = \langle MG, PF \rangle$, where MG is an SEMG and PF is a function from the mode set of MG to a set of system predicates.

$PG \text{ cmp } \langle D, SP \rangle$ Complete predicate mode graph

$PG \text{ cmp } \langle D, SP \rangle$ iff

$\forall m \in M(MG): \forall x \in MG.sr(m): \forall H \in \text{Set}(D): \forall \text{Int} \in \text{SI}(T):$

$[H \text{ sat } PF(m)@s(Int) \wedge H \text{ sat } m@Int \wedge H \text{ sat } Start(x)@e(Int) \Rightarrow H \text{ sat } PF(x)@e(Int)]$ and
 $\forall H \in \text{Set}(D): \forall t \in T: [H \text{ sat } SP@t \Rightarrow H \text{ sat } PF(S(MG))@t].$

System Specifications

Dis(SY)	<i>Disaster Set.</i> The set of system predicates that specify the disasters of system SY.
Dip(SY)	<i>Disaster Predicate.</i> The disjunction of the system predicates in the set Dip(SY).
DFH(SY)	<i>Disaster-free History Set.</i> $DFH(SY) = \{H \in \Gamma H \mid \forall x \in \text{Dis}(SY): \neg H \text{ sat } x\}.$
HS(SY)	<i>Hazard Specification.</i> A system predicate that specifies the identified hazards of system SY.
HA(SY)	<i>Hazard analysis histories</i> The set of histories that satisfy the complete hazard assumption for Dip(SY) and HS(SY)
SRD(SY)	<i>Safety Real World Description.</i> A history description that specifies the restrictions imposed on safety-critical behaviour of a system SY by the real world..
SRDH(SY)	<i>Safety Real world Description Histories.</i> $SRDH(SY) = \text{Set}(SRD(SY)).$
SRS(SY)	<i>Safety Real world Specification.</i> The negation of the hazard specification of system SY.
SRH(SY)	<i>Safety Real wold Histories.</i> $SRH(SY) = \{H \in SRDH(SY) \mid H \text{ sat } SRS(SY)\}.$
MRD(SY)	<i>Mission real world description.</i> A history description that specifies the restrictions imposed on mission-oriented behaviour of a system SY by the real world.

MRDH(SY)	<p><i>Mission Real World Histories.</i></p> <p>$MRDH(SY) = \text{Set}(MRD(SY)).$</p>
MPS(SY)	<p><i>Mission Phase Specification.</i></p> <p>A phase graph that specifies the mission requirements of SY.</p>
MRS(SY)	<p><i>Mission Real World Specification.</i></p> <p>An SEMG that specifies the mission requirements of system SY.</p>
MRH(SY)	<p><i>Mission Real World Histories.</i></p> <p>$MRH(SY) = \{H \in MRDH(SY) \mid H \text{ sat } MRS(SY)\}.$</p>
SED(SY)	<p><i>Safety Environment Description.</i></p> <p>A history description that specifies the relationship between the safety controller and safety real world variables, and the relationships of SRD(SY), of system SY.</p>
SEDH(SY)	<p><i>Safety Environment Description Histories.</i></p> <p>$SEDH(SY) = \text{Set}(SED(SY)).$</p>
SCS(SY)	<p><i>Safety Controller Specification.</i></p> <p>An SEMG that specifies the behaviour that must be exhibited by the safety controller, to ensures that the behaviour described by SRS(SY) is maintained, for system SY</p>
SCH(SY)	<p><i>Safety Controller Histories.</i></p> <p>$SCH(SY) = \{H \in SEDH(SY) \mid H \text{ sat } SCS(SY)\}$</p>
SU(SY)	<p><i>Start Up Graph.</i> SEMG that specifies the behaviour of safety controller of system SY during the start up phase.</p>
MN(SY)	<p><i>Monitor Graph.</i> SEMG that specifies the behaviour of safety controller of system SY during the monitor phase.</p>
MonInv(SY)	<p><i>Monitor Invariant.</i> $\text{MonInv}(SY) = \text{GInv}(MN(SY)).$</p>
REC(SY)(m)	<p><i>Recovery Graph.</i> SEMG that specifies the behaviour of safety controller of system SY during the recovery phase of mode m.</p>

RG(SY)(m)	<i>Reset Graph.</i> SEMG that specifies the behaviour of safety controller of system SY during the reset phase of mode m.
SH(SY)(m)	<i>Shutdown Graph.</i> SEMG that specifies the behaviour of safety controller of system SY during the shutdown phase of mode m.
EC(SY)	<i>End controller mode.</i> Unbounded mode that specifies the behaviour of safety controller of system SY during the end phase.
MED(SY)	<i>Mission Environment Description.</i> A history description that specifies the relationships for the mission controller, and the relationships of MRD(SY), of system SY.
MEDH(SY)	<i>Mission Environment Description History Set.</i> $MEDH(SY) = \text{Set}(MED(SY)).$
MCS(SY)	<i>Mission Controller Specification.</i> An SEMG that specifies the behaviour that must be exhibited by the mission controller, to ensure that the behaviour described by MRS(SY) is maintained.
MCH(SY)	<i>Mission Controller History Set.</i> $MCH(SY) = \{H \in MEDH(SY) \mid H \text{ sat } MCS(SY)\}.$

Appendix B - Real World Analysis

B.1. Introduction

In this appendix, the real world analysis of a safety-critical chemical plant is described. This case study is presented to illustrate in more detail how the guidelines of chapter 7, provide a systematic approach to the real world analysis of a system. The specifications produced at each stage of the analysis are presented. To keep the example manageable, only illustrative portions of the analysis are discussed. In particular, the case study concentrates on the safety real world analysis.

B.2. System Concept

"A (computer based) controller is required for a simple chemical plant, illustrated in figure B.1. The chemical plant must react a specified volume of chemical A with a specified volume of chemical B, to produce chemical C. After the production of C is complete a specified volume of D must be added, to produce chemical E. The operator must be able to specify the volumes of A, B and D. For the reaction of a mixture of A and B to start, the temperature of the mixture must be raised to at least $Y^{\circ}\text{K}$. For the reaction to progress efficiently the temperature of the mixture of A and B must remain at approximately $Y^{\circ}\text{K}$. For the reaction of a mixture of C and D to start, the temperature must be at least $Z^{\circ}\text{K}$; and for the reaction to progress efficiently the temperature must remain at about $Z^{\circ}\text{K}$ during the reaction. Furthermore, the product E is unstable, hence the temperature must remain at about $Z^{\circ}\text{K}$ until E is collected. The reactants are loaded via three Inlets, chemical A via InletA, chemical B via InletB and chemical D via InletD. The product E is collected via OutletE.

During the lifetime of the system a light (RLight) that indicates the status of the chemical plant, in terms of the presence or absence of a reaction, to the operator is required. The relationship between the colour of the light and the status of the reaction vessel should obey the following (informal) rules:

it should be *green* when a reaction is not in progress;

it should be *amber* when a reaction is about to start or has just started; and

it should be *red* while the reaction is in progress, or a reaction has just completed.

The mission (or safety) operator interacts with the controller via a selector at the operator console (Plant select) – to specify the sequence in which reactions are to be performed in the reaction vessel; and when the product should be collected.

Furthermore, the chemicals A, B, C, D and E are known to be hazardous, hence the chemical plant must be certified by a licensing authority. The safety subsystem must ensure that the overall chemical plant will not enter into a hazardous state. The safety operator interacts with safety controller via a selector at the operator console (Safety select). OutletS is a safety outlet which is used to empty the vessel. During the lifetime of the system a light (SafeLight) indicating the safety status of the chemical plant is required. The relationship between the colour of the light and the status of the reaction vessel should obey the following (informal) rules:

it should be *green* when a reaction is not in progress;

it should be *amber* when a reaction may be in progress; and

it should be *red* when the safety controller must override the mission controller to prevent the system entering into a hazardous state.” This chemical plant will be referred to as CP for brevity.

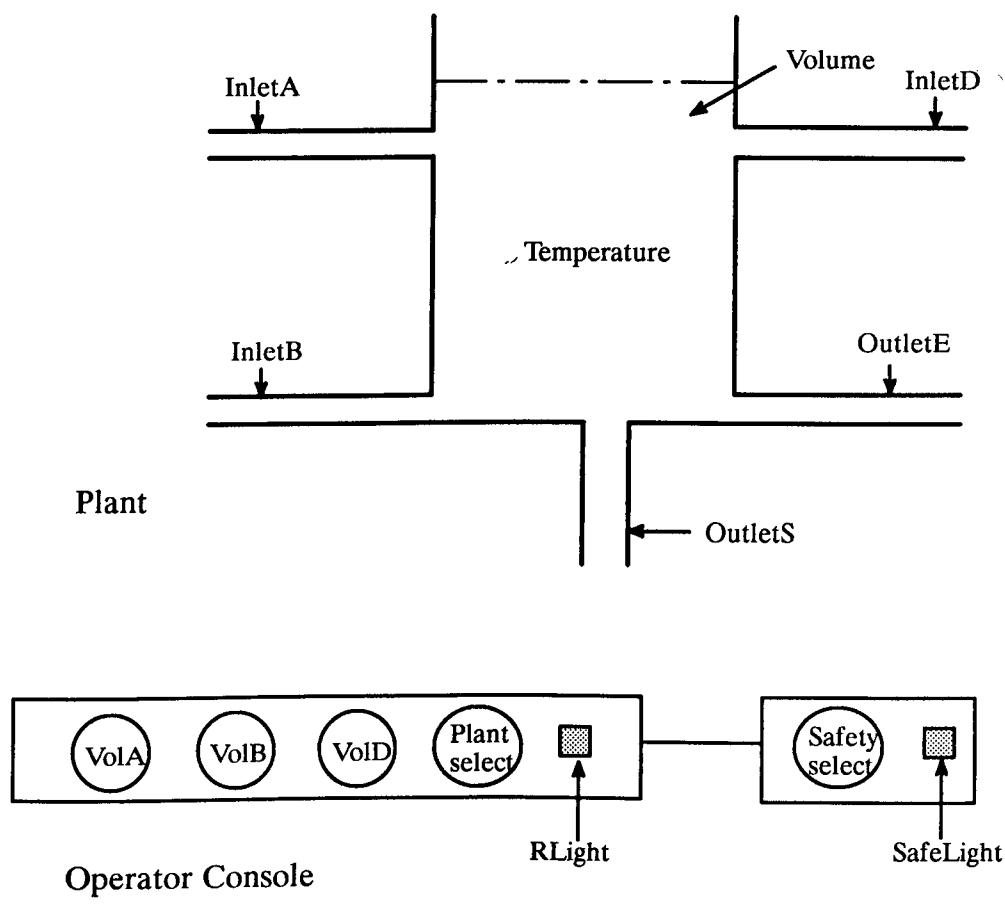


Figure B.1. Chemical Plant

B.3. Initial Real World Description Analysis.

The first stage of the real world analysis is the identification of an initial real world description for the chemical plant. The description produced as a result of this analysis, is not verified since it provides a basis for further analysis – it is not a safety-critical specification. The analysis of the real world variables of the chemical plant, by following the guidelines of section 7.2.1 is presented below.

Step 1

The time base will be measured in seconds.

Step 2

In this step the state variables of the chemical plant are identified.

The system concept contains the phrase “volume of chemical A”, in the formal model we capture this notion by the introduction of the state variable p_2 . This state variable represents the volume of chemical A in the vessel”, we give this the name *VolA*; and the units dm^3 . A similar analysis identifies and describes the state variables: *VolB*, *VolC*, *VolD*, *VolE*, *Volume*, *Temperature*, *RLight*, *Plant select*, *SafeLight*, *Safety select*, *VolA select*, *VolB select* and *VolD select*.

The system concept contains the phrase “ reactants are loaded via three Inlets, chemical A via Inlet A”, in the formal model we capture this notion by the introduction of the state variable p_{16} . This variable represents the rate at which chemical A flows into the vessel (via Inlet A), we give this the name *FlowA* and the units dm^3/s . A similar analysis identifies and describes the state variables: *FlowB*, *FlowD*, *Flow*, *OutFlowE*, *OutFlowS* and *Outflow*.

The variables are recorded in table B.1

Table B.1: Real World State Variables

Notation	Units	Name	Comments
p_1	seconds (s)	Clock	The perfect clock of the system.
p_2	dm^3	VolA	The volume of chemical A in the vessel.
p_3	dm^3	VolB	The volume of chemical B in the vessel.
p_4	dm^3	VolC	The volume of chemical C in the vessel.
p_5	dm^3	VolD	The volume of chemical D in the vessel.
p_6	dm^3	VolE	The volume of chemical E in the vessel.
p_7	dm^3	Volume	The volume of chemicals in the vessel.
p_8	$^{\circ}K$	Temperature	The temperature of the liquid in the vessel.
p_9	dm^3	VolA select	The selector for the volume of A.
p_{10}	dm^3	VolB select	The selector for the volume of B.
p_{11}	dm^3	VolD select	The selector for the volume of D.
p_{12}	Op_setting	Plant select	The selector that allows the operator to control the sequence of operations of the mission controller.
p_{13}	Sa_setting	Safety select	The selector that allows the safety operator to control the sequence of operations of the safety controller.
p_{14}	Colour	RLight	The indicator which informs the operator of the current status of the plant.
p_{15}	Colour	SafeLight	The indicator which informs the safety operator of the current safety status of the plant.
p_{16}	dm^3/s	FlowA	The flow rate of chemical A into the vessel.
p_{17}	dm^3/s	FlowB	The flow rate of chemical B into the vessel.

P18	dm ³ /s	FlowD	The flow rate of chemical D into the vessel.
P19	dm ³ /s	Flow	The combined flow rate of chemicals into the vessel.
P20	dm ³ /s	OutFlowE	The flow rate of chemicals out of the vessel via OutletC.
P21	dm ³ /s	OutFlowS	The flow rate of chemicals out of the vessel via OutletD.
P22	dm ³ /s	Outflow	The combined flow rate of chemicals out of the vessel.

Step 3

The ranges of the variables are defined by systematically analyzing the variables. For example, let us suppose that an analysis of the selector represented by VolA select, shows that the minimum value that selector can be used to select is V_l, the granularity of the selector is Δv and the selector can be used to select V + 1 different values. The range of VolA is given by the set V_{p9} = {V_l, V_l + Δv, , ... , V_l + V.Δv}. A similar analysis is performed over the rest of the variables, the results of this analysis are recorded in Table B.2.

Table B.2: Real World Variable Ranges

Variables	Range	Comments
P1	T	The range of the clock is the systems lifetime.
P2, ..., P7	{x ∈ R 0 ≤ x ≤ Vmax}	Vmax is the maximum volume of liquid that the vessel can contain.
P8	{x ∈ R T _l ≤ x ≤ T _u }	T _l and T _u are the upper and lower limits of the temperature of the contents of the vessel.
P9, P10, P11	{V _l , V _l + Δv, , ... , V _l + V.Δv}	V _l is the minimum value that can be selected, Δv is a number (Δv > 0) that represents the granularity of the selector and V _l + V.Δv represents the maximum value.
P12	{off, on, start, collect}	Where off, on, start and collect are the states of Plant select.
P13	{off, on}	Where off and on, are the states of Safety select
P14, P15	{g, a, r}	Where g: green, a: amber and r: red.
P16	{x ∈ R 0 ≤ x ≤ FmaxA}	FmaxA is the maximum flow rate of chemical A into the vessel.
P17	{x ∈ R 0 ≤ x ≤ FmaxB}	FmaxB is the maximum flow rate of chemical B into the vessel.
P18	{x ∈ R 0 ≤ x ≤ FmaxD}	FmaxD is the maximum flow rate of chemical D into the vessel.
P19	{x ∈ R 0 ≤ x ≤ Fmax}	Fmax is the maximum flow rate of chemicals into the vessel.
P20	{x ∈ R 0 ≤ x ≤ OmaxE}	OmaxE is the maximum flow rate of chemicals out of the vessel, via OutletE.
P21	{x ∈ R 0 ≤ x ≤ OmaxS}	OmaxS is the maximum flow rate of chemicals out of the vessel, via OutletS.
P22	{x ∈ R 0 ≤ x ≤ Omax}	Omax is the maximum flow rate of chemicals out of the vessel.

Step 4

The categories and classes are determined by a systematic analysis of the variables. For example, consider the following (typical) analysis of the variable Plant select. Since the variable represents part of the state of the operator console it is clearly an Operator variable. Let us suppose that an analysis of the selector represented by Plant select, shows that this selector consists of a set of buttons (one for each state); the state of the selector being given by the depressed button. Further let us suppose that the buttons, can be pressed in any order. Hence the class of Plant select is free. A similar analysis is performed over the other state variables, the results of this analysis are recorded in Table B.3.

Table B.3: Real World Variable Categories and Classes

Variables	Category	Class	Comments
p ₁	Physical	Perfect clock	The variable p ₁ represents the perfect clock of the system.
p ₂ , ..., p ₈ p ₁₆ , ..., p ₂₂	Physical	Continuous	The variables which represent the physical properties of the reaction vessel have continuous restricted histories.
p ₉ , ..., p ₁₅	Operator	Free	No restrictions are imposed on the behaviour of the selectors.

Step 5

In this step we identify the invariant relations.

a. In this step we investigate variables of the same units (dimensions) to identify invariant reactions.

There are three types of units, that are common to more than one state variable these are: dm^3 , *Colour* and dm^3/s . Next we present the three groups; and identify relations over them, by analyzing the relationship between the variables of the group in the context of the construction of the plant.

Volume variables (dm^3). The group of volume variables are: p₂, ..., p₇, p₉, p₁₀, p₁₁.

The variables p₉, p₁₀ and p₁₁ represent the state of the volume selectors, let us suppose that an analysis of the plant shows that no invariant relation is imposed over them. The variables p₂, ..., p₇, represent the volumes of the different chemicals in the vessel. Let us

suppose that any intermediate products from the reaction of A and B can be regarded as C; and any intermediate products from the reaction of C and D can be regarded as E. Hence, since the vessel can contain only A, B, C, D or E, at any time the volume of chemical in the vessel is the sum of the individual volumes. This is captured by the invariant relation:

$$p_7 = p_2 + p_3 + p_4 + p_5 + p_6.$$

Light variables (colour). The group of light variables are: p_{14} and p_{15} . Let us suppose that an analysis shows that no relation is imposed over them.

Flow variables (dm^3/s). The flow variables are: p_{16} , ..., p_{22} .

Since there are only three inlets into the reaction vessel the total flow rate into the vessel is the sum of the flow rates through the three inlets. This is captured by the invariant relation:

$$p_{19} = p_{16} + p_{17} + p_{18}. \text{ Similarly we can derive the following invariant relation for the OutFlow: } p_{22} = p_{20} + p_{21}.$$

b. In this step we investigate variables of the same class to identify invariant relations. Let us suppose that the analysis over the variables in the two classes Continuous and Free does not identify any new invariant relations.

c. In this step we investigate any physical properties of the application.

Let us suppose that a property of the reactions is that there is no change in volume caused by a reaction. The consequences of this property cannot be expressed by an invariant relation, since its effect is over an interval.

d. Let us suppose that an analysis of the state of the real world variables, at the start point of the system lifetime, identifies the fact that the vessel is empty and the temperature below Stemp (see Ir₄).

Step 6

a. In this step we investigate variables of the same class to identify history relations.

As an example consider the Continuous class, this contains the variables p_2 , ..., p_8 and p_{16} , ..., p_{22} .

An analysis of the variable *VolA* and the construction of the plant shows that its value is influenced by *FlowA*. This influence can be stated informally as, for any interval the value

of *VolA* at the end of the interval is bounded by the sum of *VolA* at the start of the interval and the integral of the *FlowA* during the interval. This relation can be stated as: $p_{2,1} \leq p_{2,0} + \int p_{16}(t) dt$. Similar relations can be formulated for the volumes of B and D.

b. In this step we investigate relations over the derivatives of a variable.

Let us suppose that an analysis shows that the only relevant relation is an upper bound on the rise of the temperature. Hence we conclude the history relation: $p_{8,1} \leq p_{8,0} - \Delta T m.dur$.

c. In this step we investigate properties identified in step 5.c, that were not expressed as invariant relations.

Recall that we identified the property “that no change in volume caused by a reaction”. The main consequences of this are that changes in volume depend only on flow rates into and out of the vessel; and the maximum volume of the vessel. Hence we conclude the history relation: $p_{7,1} = \min(p_{7,0} + \int (p_{19}(t) - p_{22}(t)) dt, V_{max})$.

The relations are summarised in table B.4. It should be noted that this table represents only an initial attempt at capturing the real world properties (some of the relations may change during later stages of the analysis as the understanding of the system improves).

Table B.4: Relations of Initial Real World Description

No.	Related variables	Relationship	Comments
Ir ₁	p ₂ , p ₃ , p ₄ , p ₅ , p ₆ , p ₇	$p_7 = p_2 + p_3 + p_4 + p_5 + p_6$	The volume of liquid in the vessel is the sum of the volumes of A, B, C, D and E.
Ir ₂	p ₁₆ , p ₁₇ , p ₁₈ , p ₁₉	$p_{19} = p_{16} + p_{17} + p_{18}$	The flow rate into the vessel is the sum of the flow rates of A, B and D.
Ir ₃	p ₂₀ , p ₂₁ , p ₂₂	$p_{22} = p_{20} + p_{21}$	The flow rate out of the vessel is the sum of the flow rates out of OutletE and OutletS.
Ir ₄	p ₁ , p ₇ , p ₈	$p_1 = S(T) \Rightarrow p_7 = 0 \wedge p_8 < Stemp$	At the start of the system lifetime there are no cheimcals in the vessel and the temperature is below Stemp.
Hr ₁	p ₇ , p ₁₉ , p ₂₂	$p_{7,1} = \min(p_{7,0} + \int (p_{19}(t) - p_{22}(t))dt, V_{max})$	The volume of liquid in the vessel at the end of any interval is the smaller of V _{max} and the sum of the volume at the start of the interval and the integral of the difference of the flow rate into the vessel and out of the vessel during that interval.

Hr ₂	p ₂ , p ₁₆	$p_{2,1} \leq p_{2,0} + \int p_{16}(t) dt$	The volume of A at the end of any interval is at most the sum of the volume of A at the start of the interval and the integral of the the flow rate during the interval.
Hr ₃	p ₃ , p ₁₇	$p_{3,1} \leq p_{3,0} + \int p_{17}(t) dt$	The equivalent relation to Hr ₂ for B.
Hr ₄	p ₅ , p ₁₈	$p_{5,1} \leq p_{5,0} + \int p_{17}(t) dt$	The equivalent relation to Hr ₂ for D.
Hr ₅	p ₈	$p_{8,1} \leq p_{8,0} + \Delta T_m \times dur$	ΔT_m is the maximum rise in the temperature per second, during any interval.

Step 7

The initial real world description of the chemical plant is given as:

$IRWD(CP) = \langle T, \langle p_1, ..., p_{22} \rangle, \langle V_{p1}, ..., V_{p22} \rangle, \langle C_{p1}, ..., C_{p22} \rangle, \langle Ir_1, Ir_2, Ir_3, Ir_4 \rangle, \langle Hr_1, ..., Hr_5 \rangle \rangle.$

B.4. Safety Real World Analysis

In this section we show, how SRD(CP) and SRS(CP) are produced by performing a disaster analysis, hazard analysis, safety real world description analysis and a safety real world specification analysis.

B.4.1 Disaster Analysis

In this section we show how Dis(CP), is produced by following the guidelines of section 7.3.

Disaster Identification

Step 1

Let us suppose that the disaster analysts of the chemical plant select a suitable check-list.

Step 2

Let us suppose that the disaster analysts analyse the check-list of step 1, and record the results in a disaster analysis table. An extract of this disaster analysis table, that shows the two potential disasters of the chemical plant is given below (table B.5).

Table B.5: Disaster Analysis of Chemical Plant

Possible Disasters	Results
Toxic vapour or fluid release	The release of toxic vapours is a potential disaster, the occurrence of which is denoted by the predicate p ₂₃ .

Explosion	An explosion is a potential disaster, the occurrence of which is denoted by the predicate p_{24} .
Personnel exposure to radiation.	This is not a possible disaster, since no radioactive materials are used in the chemical plant.

Step 3

The state variables, that represent the disasters are described in the following table.

Table B.6: Real World State Variables

Notation	Units	Name	Comments
p_{23}	Tx_rating	Toxic	The occurrence or non-occurrence of the release of toxic fumes.
p_{24}	Ex_rating	Explosion	The occurrence or non-occurrence of an explosion.

The range of *Toxic* and *Explosion* is the set $\{true, false\}$, their category Physical and their class the *Catastrophe* class.

Validation

For simplicity we will assume that an independent analysis has been conducted which identifies the same disaster analysis table. Hence the disaster set of the chemical plant is simply: $Dis(CP) = \{p_{23}, p_{24}\}$; and the disaster predicate $Dip(CP) = p_{23} \vee p_{24}$.

B.4.2. Hazard Specification Analysis

In this section we show how $HS(CP)$ is produced by following the guidelines of section 7.4.

Hazard Identification

Step 1

In this step we perform a hazard analysis, over the real world variables of the chemical plant. The hazards of the chemical plant are identified by a systematic analysis of the variables to identify the system conditions (if any) involving each variable that can lead to a disaster. The hazard analysis table of the chemical plant (produced by the systematic analysis of the real world variables) is given in table B.7.

Table B.7: Hazard Analysis of Chemical Plant

Variables	Comments	Hazards
p_2, p_8	An explosion, can occur if there is some A in the vessel and the temperature is above $Eact_A$.	$p_2 \neq 0 \wedge p_8 > Eact_A$.

p3, p8	An explosion, can occur if there is some B in the vessel and the temperature is above Eact _B .	$p_3 \neq 0 \wedge p_8 > \text{Eact}_B$
p4, p8	An explosion can occur if there is some C in the vessel and the temperature is above Eact _C .	$p_4 \neq 0 \wedge p_8 > \text{Eact}_C$
p5, p8	An explosion can occur if there is some D in the vessel and the temperature is above Eact _D .	$p_5 \neq 0 \wedge p_8 > \text{Eact}_D$
p6, p8	Toxic fumes are released from the vessel if the volume of E in the vessel is greater than Tvol and the temperature is above Tact.	$p_6 > \text{Tvol} \wedge p_8 > \text{Eact}$
p7, p9, ..., p22	In the absence of the conditions above, these variables do not define any hazards	

Remark: For simplicity we have assumed that hazards can only occur, while there are some chemicals in the reaction vessel. That is, for the purposes of this example, we have ignored the possibility of hazards while the reactants are stored or the product(s) are collected.

Step 2

By the inspection of table B.7, we construct the hazard predicates for the release of toxic fumes and the occurrence of an explosion. These hazard specifications are then used to derive the hazard specification of the chemical plant.

The hazardous states for release of toxic fumes are captured by the system predicate:

$$\text{HZ}(p_{23}) = (p_6 > \text{Tvol} \wedge p_8 > \text{Tact}).$$

The hazardous states for an explosion are captured by the system predicate:

$$\begin{aligned} \text{HZ}(p_{24}) = & (p_2 \neq 0 \wedge p_8 > \text{Eact}_A) \vee (p_3 \neq 0 \wedge p_8 > \text{Eact}_B) \vee (p_4 \neq 0 \wedge p_8 > \text{Eact}_C) \vee \\ & (p_5 \neq 0 \wedge p_8 > \text{Eact}_D). \end{aligned}$$

Step 3

The release of toxic fumes and an explosion are the only two disasters of the chemical plant (see disaster analysis). Hence the hazard specification of the chemical plant is:

$$\text{HS}(\text{CP}) = \text{HZ}(p_{23}) \vee \text{HZ}(p_{24}).$$

$$\begin{aligned} \text{HS}(\text{CP}) = & (p_6 > \text{Tvol} \wedge p_8 > \text{Tact}) \vee (p_2 \neq 0 \wedge p_8 > \text{Eact}_A) \vee (p_3 \neq 0 \wedge p_8 > \text{Eact}_B) \\ & \vee (p_4 \neq 0 \wedge p_8 > \text{Eact}_C) \vee (p_5 \neq 0 \wedge p_8 > \text{Eact}_D). \end{aligned}$$

Validation Guidelines

For simplicity we will assume that an independent analysis has been conducted which identifies the same hazard analysis table.

Hazard Elimination

Let us suppose that a safer (economical) process route does not exist for the production of E.

Complete Hazard Assumption.

Let us suppose that the hazard analysts confirm that for any possible history H of the system, the following holds: $\forall t \in T: [H \text{ sat } p_{23} \vee p_{24} @ t \Rightarrow \exists t' \in [S(T), t): H \text{ sat } HS(CP) @ t']$. Roughly speaking, if a history H satisfies the above condition then a disaster can occur at a time point t during H if and only if H satisfies the hazard specification at some time point before t . We define the set of hazard analysed histories as:

$$HA(CP) = \{H \in \Gamma H \mid \forall t \in T: [H \text{ sat } p_{23} \vee p_{24} @ t \Rightarrow \exists t' \in [S(T), t): H \text{ sat } HS(CP) @ t']\}.$$

Where ΓH is set of all functions $H: T \rightarrow V_{p_1} \times \dots \times V_{p_{24}}$.

From lemma 5.1 we infer that: $\forall H \in HA(CP): H \text{ sat } \neg HS(CP) \Rightarrow H \text{ sat } \neg(p_{23} \vee p_{24})$.

B.4.3. Safety Real world Description

In this section we show how $SRD(CP)$ is produced by following the guidelines of section 7.5.

Construction Guidelines

Step 1

In this step we identify the safety real world variables of the chemical plant.

- a. From the hazard specification and disaster set we can define the initial set of safety real world variables as: $p_2, p_3, p_4, p_5, p_6, p_8, p_{23}$ and p_{24} .
- b. We investigate the relations of $IRWD(CP)$, to identify the variables related to those identified in *step 1.a*.

From invariant relation Ir_1 we identify p_7 , and from history relations Hr_2, Hr_3 and Hr_4 we identify p_{16}, p_{17} and p_{18} .

Next we investigate the relations of IRWD to identify those related to p_7 , p_{16} , p_{17} and p_{18} . From history relation Hr_1 we identify p_{19} and p_{22} ; and from invariant relation Ir_3 we identify p_{19} .

Hence the set of safety real world variables is: $\{p_2, \dots, p_8, p_{16}, \dots, p_{22}\}$.

Step 2 (Invariant relations)

- a. The invariant relations (of IRWD(CP)) involving the safety real world variables are: Ir_1 , Ir_2 , Ir_3 and Ir_4 .
- b. Let us suppose an analysis of the safety real world variables shows that their are no other relevant relations.

Step 3 (History relations)

- a. The history relations (of IRWD(CP)) involving the safety real world variables are: Hr_1 , ..., Hr_5 .
- b. Let us suppose that an analysis of the reaction between A and B, shows that to produce $2x$ dm^3 in the vessel, the vessel must contain at least x dm^3 of B. Hence we can infer the following relation (Hr_6): $p_{4,1} + 2.p_{3,1} \leq p_{4,0} + 2.p_{3,0} + \int p_{17}(t) dt$. Let us suppose that a similar relation is defined for the reaction involving C and D (see Hr_7).

The description relations are summarized in table B.8.

Table B.7: Additional Relations of Safety Real World Description

No.	Related variables	Relationship	Comments
Hr6	p_3, p_4, p_{17}	$p_{4,1} + 2.p_{3,1} \leq p_{4,0} + 2.p_{3,0} + \int p_{17}(t) dt$	For any interval Int , the sum of VolC and twice VolB at $e(Int)$ is at most the sum of VolC and twice VolB at $s(Int)$ and the integral of FlowB during Int .
Hr7	p_5, p_6, p_{18}	$p_{6,1} + 2.p_{5,1} \leq p_{6,0} + 2.p_{5,0} + \int p_{18}(t) dt$	The equivalent relation to Hr_6 for VolD, VolE and FlowD.

Step 4

The safety real world description of the chemical plant is given as:

$SRD(CP) = \langle T, \langle p_1, \dots, p_{24} \rangle, \langle V_{p_1}, \dots, V_{p_{24}} \rangle, \langle C_{p_1}, \dots, C_{p_{24}} \rangle, \langle Ir_1, Ir_2, Ir_3, Ir_4 \rangle, \langle Hr_1, \dots, Hr_7 \rangle \rangle$.

Validation Guidelines

Again we assume that SRD(CP) is validated by an independent analysis.

B.4.4. Safety Real World Specification

In this section we show how SRS(CP) is produced by following the guidelines of section 7.6.

Step 1

The safety real world specification is defined as the negation of HS(CP).

$$\begin{aligned}
 \text{SRS(CP)} &= \neg \text{HS(CP)}. \\
 &= \neg [(p_6 > \text{Tvol} \wedge p_8 > \text{Tact}) \vee (p_2 \neq 0 \wedge p_8 > \text{Eact}_A) \vee (p_3 \neq 0 \wedge p_8 > \text{Eact}_B) \\
 &\quad \vee (p_4 \neq 0 \wedge p_8 > \text{Eact}_C) \vee (p_5 = 0 \wedge p_8 > \text{Eact}_D)]. \\
 &= [(p_6 \leq \text{Tvol} \vee p_8 \leq \text{Tact}) \wedge (p_2 = 0 \vee p_8 \leq \text{Eact}_A) \wedge (p_3 = 0 \vee p_8 \leq \text{Eact}_B) \\
 &\quad \vee (p_4 = 0 \vee p_8 \leq \text{Eact}_C) \wedge (p_5 = 0 \vee p_8 \leq \text{Eact}_D)].
 \end{aligned}$$

Step 2

We must check that a history that satisfies SRS(CP) may also satisfy the mission. Let us suppose that Eact_C is less than Eact_A , Eact_B and Eact_D , and Eact_C is greater than Y . Further let us suppose that Tact is greater than Z , but less than Y .

SRS(CP) will allow the reaction of A and B (provided there is no C in the vessel); and allow the reaction of C and D.

Step 3

SRS(CP) is quite complex, here we investigate if the relations of SRD(CP) can be used to define a simpler version.

Consider the following system predicate:

$$\text{SC} = [(p_6 \leq \text{Tvol} \vee p_8 \leq \text{Tact}) \wedge ((p_2 = 0 \wedge p_3 = 0 \wedge p_4 = 0 \wedge p_5 = 0) \vee p_8 \leq \text{Eact}_C)].$$

We can say for any history H of SRDH, $H \text{ sat SC} \Rightarrow H \text{ sat SRS(CP)}$ since $\text{Eact}_C < \text{Eact}_A$, $\text{Eact}_C < \text{Eact}_B$ and $\text{Eact}_C < \text{Eact}_D$.

Consider the system predicate $(p_2 = 0 \wedge p_3 = 0 \wedge p_4 = 0 \wedge p_5 = 0)$, the relation Ir_1 states that the $p_7 = p_2 + p_3 + p_4 + p_5 + p_6$. Hence, for any history H of SRDH:

$$\forall t \in T: H \text{ sat } (p_7 = 0 \Rightarrow p_2 = 0 \wedge p_3 = 0 \wedge p_4 = 0 \wedge p_5 = 0) @ t.$$

Hence, we simplify SRS(CP) to: $(p_6 \leq \text{Tvol} \vee p_8 \leq \text{Tact}) \wedge (p_7 = 0 \vee p_8 \leq \text{Eact}_C)$.

Hereafter, we will refer to Eact_C as Eact .

B.5. Mission Real World Analysis

In addition to the requirements given by the system concept the customer adds the requirement that a hazard should not occur for any behaviour that satisfies the mission. That is, the safety controller need not, in fact, need to override the mission controller, if the mission controller ensures that the behaviour at the real world complies to the mission real world specification. In this section we present an overview of how MRS(CP) and MRD(CP) are produced by following the guidelines of sections 7.7, 7.8 and 7.9.

B.5.1. Mission Phase Specification

In this section, we present an overview of the analysis performed over the system concept to construct MPS(CP). By following the guidelines of section 7.7.1.

High-Level Phase Analysis

The analysis starts with the standard high-level phase graph (shown in figure B.2).



Figure B.2. Chemical Plant High-level Phase Graph

Select phase

At the start of this phase, the vessel will be empty. During this phase the vessel must remain empty. By the end of the phase the operator must have selected the required volumes of A, B and D.

Produce phase

At the start of this phase, the required volumes of A, B and D will be selected. During this phase the system must not be in a hazardous state. By the end of the phase the plant must have produced the chemical E.

Collect phase

At the start of this phase, chemical E must have been produced. During this phase the

system must not be in a hazardous state. By the end of the phase, the collection of chemical E, must have been completed.

Phase Analysis

For the chemical plant, it is not necessary to decompose the high-level phases any further.

Mission Phase Specification Check

The mission phase graph is presented to the customer, who raises the following (new) point to the analyst: “The system should be able to perform a sequence of reactions”.

To allow multiple reactions during the system lifetime, the analyst modifies the phase graph, to that given in figure. B.3. The specification of the new phases is also refined by introducing the real world variables.

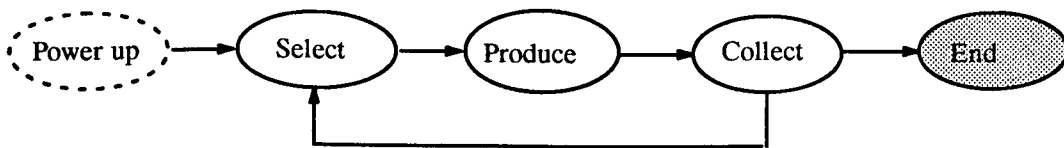


Figure B.3. Chemical Plant Mission Phase Specification

Power up phase

At the start of this phase the vessel will be empty. While the system is in the phase the vessel is empty and RLight is at green. The system leaves the phase as soon as Plant select is at on.

Select phase

At the start of this phase the vessel will be empty and Plant select at on. While the system is in this phase the vessel is empty and Plant select is at on or start and RLight is at green. The system leaves the phase as soon as the requested volumes of A, B and D have been selected.

Produce phase. As before.

Collect phase

While the system is in this phase the temperature must be approximately at Z; and RLight at amber. The system leaves this phase as soon as the product E has been collected.

End phase

While the system is in this phase the vessel is empty and the indicator at green. The system remains in this phase until the system is shut down.

Mission Real World Specification Analysis

In this section we present an overview of how MRS(CP) is produced by following the guidelines of section 7.7.2.

$$BF(\text{Power up}) = \text{Power on}$$

Power on mode

The system starts in this mode. At the start of this mode Plant select is at off. During this mode, the vessel must remain empty, Plant select at off or on and RLight is green. The system leaves this mode as soon as Plant select is at on.

$$\text{Power on} = \langle p_{12} = \text{off}, p_7 = 0 \wedge p_{12} \in \{\text{off}, \text{on}\} \wedge p_{14} = g, p_{12} = \text{on} \rangle.$$

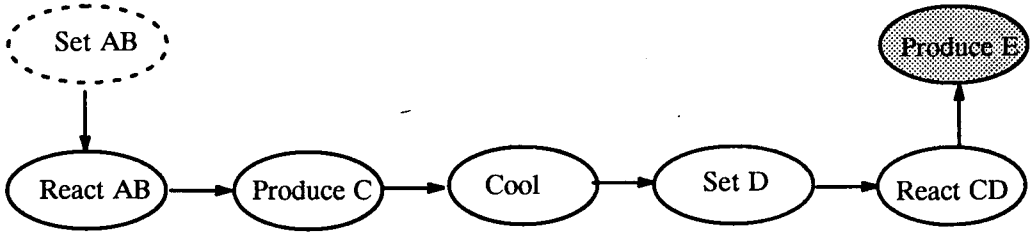
$$BF(\text{Select}) = \text{Select Vol}$$

Select Vol mode

The system is in this mode while the operator selects the required volumes of A, B and D. At the start of this mode Plant select is at on. During this mode, the vessel must be empty, Plant select at on or start and RLight is green. The system leaves this mode as soon as the operator has selected the required volumes, which he signals by setting plant select to start.

$$\text{Select vol} = \langle p_{12} = \text{on}, p_7 = 0 \wedge p_{12} \in \{\text{on}, \text{start}\} \wedge p_{14} = g, p_{12} = \text{start} \rangle.$$

BF(Produce) =



Set AB mode

The system is in this mode while the requested volumes of A and B are being loaded into the vessel. At the start of this mode the vessel is empty. During this mode, there is no D or E in the vessel, the Temperature is less than $Y + \Delta R$, Plant select is at start, RLight is green or amber. The system leaves this mode as soon as the volumes of A and B are approximately at the requested values and RLight is amber; and chemical flow into and out of the vessel is zero. The system must spend at most $Su(p_9, p_{10})$ seconds in this mode, where Su defines an upper bound on the time required to load the requested volumes of A and B into the vessel.

Set AB =

$\langle p_7 = 0, ZDE \wedge p_8 \leq Y + \Delta R \wedge p_{12} = \text{start} \wedge p_{14} \in \{g, a\}, RS \wedge p_{14} = a \wedge ZF, 0, Su(p_9, p_{10}) \rangle$,

where $ZDE = p_5 = 0 \wedge p_6 = 0$;

$RS = (|p_2 - p_9| \leq \Delta V_A \wedge |p_3 - p_{10}| \leq \Delta V_B)$; and

$ZF = (p_{19} = 0 \wedge p_{22} = 0)$.

React AB mode

The system is in this mode while the temperature of the mixture of A and B is being raised to Y, to initiate the reaction. At the start of this mode the Temperature is less than Y and RLight is at amber. During this mode, there is no D or E in the vessel and the Temperature is at most $Y + \Delta R$, Plant select is at start, RLight is amber or red, and chemical flow into and out of the vessel is zero. The system must leave this mode as soon as the Temperature is in the range Y to $Y + \Delta R$ and RLight is red. The system must spend at most Ru seconds in this mode, where Ru is an upper bound on the time required to initiate a reaction between A and B.

React AB =

$\langle p_8 < Y \wedge p_{14} = a, ZDE \wedge p_8 \leq Y + \Delta R \wedge p_{12} = \text{start} \wedge p_{14} \in \{a, r\} \wedge ZF, RY \wedge p_{14} = r, 0, Ru \rangle$,
 where $RY = (Y \leq p_8 \leq Y + \Delta R)$.

Produce C mode

The system is in this mode while the reaction involving A and B is in progress. During this mode, there is no D or E in the vessel, the Temperature is in the range Y to $Y + \Delta R$, Plant select is at start, RLight is amber or red and chemical flow into and out of the vessel is zero. The system leaves this mode as soon as the reaction is complete and RLight is set to amber. The system must spend between $CL(p_9, p_{10})$ and $CL(p_9, p_{10}) + \Delta C$ seconds in the mode. Where CL defines an upper bound on the time needed for reaction of the required volumes of A and B to be completed.

Produce C =

$\langle \text{true}, ZDE \wedge RY \wedge p_{12} = \text{start} \wedge p_{14} \in \{a, r\} \wedge ZF, p_{14} = a, CL(p_9, p_{10}), CL(p_9, p_{10}) + \Delta C \rangle$.

Cool mode

The system is in this mode while the temperature of the product C is cooled below Z. At the start of this mode the Temperature is in the range Y to $Y + \Delta R$. During this mode, there is no D or E in the vessel the Temperature is at most $Y + \Delta R$, Plant select is at start, and RLight is green or amber, and chemical flow into and out of the vessel is zero. The system leaves this mode as soon as the Temperature is below Z and RLight is green. The system must spend at most C_u seconds in this mode, where C_u is a fixed maximum upper bound on the time required to cool C after it has been produced by the reaction of A and B.

$\text{Cool} = \langle RY, ZDE \wedge p_8 \leq Y + \Delta R \wedge p_{12} = \text{start} \wedge p_{14} \in \{g, a\} \wedge ZF, p_8 < Z \wedge p_{14} = g, 0, C_u \rangle$.

Set D

The system is in this mode while the required volume of D is being entered into the vessel. At the start of this mode there is no D in the vessel. During this mode, the temperature is below Z, Plant select is at start, RLight is green or amber, no A or B flows into the vessel, and no chemical flows out of the vessel. The system leaves this mode as soon as the vessel contains approximately the required volume of D and RLight is amber. Furthermore, the system must not spend more than $SDu(p_{11})$ seconds in this mode, where SDu is defines an upper bound on

the time required to load $p_{11} \text{ dm}^3$ of D .

Set $D = \langle p_5 = 0, \text{Inv}, |p_5 - p_{11}| \leq \Delta D_v \wedge p_{14} = a, 0, \text{SDu}(p_{11}) \rangle$,

where $\text{Inv} = (p_8 < Z \wedge p_{12} = \text{start} \wedge p_{14} \in \{g, a\} \wedge p_{16} = 0 \wedge p_{17} = 0 \wedge p_{22} = 0)$.

React CD

The system is in this mode while the temperature of the mixture of C and D is being raised to Z . At the start of this mode the volume of D is approximately the required volume and RLight is amber. During this mode, the Temperature is below $Z + \Delta R$, Plant select is at start, RLight is amber or red, and chemical flow into and out of the vessel is zero. The system leaves this mode as soon as the Temperature is in the range Z to $Z + \Delta R$ and RLight is red. Furthermore, the system must not spend more than R_u seconds in this mode, where R_u is an upper bound on the time required to initiate the reaction between C and D .

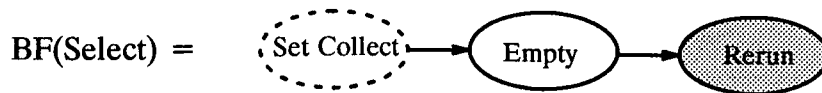
React $CD = \langle |p_5 - p_{11}| \leq \Delta D_v \wedge p_{14} = a, \text{Inv}, RZ \wedge p_{14} = r, 0, R_u \rangle$,

where $RZ = (Z \leq p_8 \leq Z + \Delta R)$ and $\text{Inv} = (p_8 \leq Z + \Delta R \wedge p_{12} = \text{start} \wedge p_{14} \in \{a, r\} \wedge ZF)$.

Produce E

The system is in this mode while the reaction involving C and D is in progress. During this mode, the temperature is in the range Z to $Z + \Delta R$, Plant select is at start, and RLight is green or red, and chemical flow into and out of the vessel is zero. The system leaves this mode as soon as the reaction has completed and RLight is at red. The system must spend between $DL(p_{11})$ and $DL(p_{11}) + \Delta D$ seconds in the mode. Where DL defines an upper bound on the time needed for reaction of the required volume of D to be completed.

Produce $E = \langle \text{true}, RZ \wedge p_{12} = \text{start} \wedge p_{14} \in \{g, r\} \wedge ZF, p_{14} = r, DL(p_{11}), DL(p_{11}) + \Delta D \rangle$.



Set Collect

The system is in this mode while the operator sets plant select to collect. During this mode the Temperature is in the range Z to $Z + \Delta R$, Plant select is at start or collect, RLight is green, and chemical flow into and out of the vessel is zero. The system leaves this mode as soon as Plant

select is at collect.

Set Collect = $\langle \text{true}, RZ \wedge p_{12} \in \{\text{start}, \text{collect}\} \wedge p_{14} = g \wedge ZF, p_{12} = \text{collect} \rangle$.

Empty

The system is in this mode while the final product (i.e., E) is being collected. During this mode the Temperature is in the range Z to $Z + \Delta R$, Plant select is at collect, RLight is at green, chemical flow into the vessel is zero, and the only outflow is via OutletC. The system leaves this mode as soon as the product has been collected (i.e., vessel is empty). Furthermore, the system must not spend more than E_u seconds in this mode, where E_u defines a maximum upper bound on the time required to empty the vessel.

Empty = $\langle \text{true}, RZ \wedge p_{12} = \text{collect} \wedge p_{14} = g \wedge p_{19} = 0 \wedge p_{22} = p_{21}, p_7 = 0, 0, E_u \rangle$.

Rerun

The system is in this mode while the operator decides if another batch of C will be produced or the system will be shutdown. During this mode the vessel is empty, Plant select is at off, on or collect, and RLight is green. The system leaves this mode as soon as Plant select is at off or on.

Rerun = $\langle \text{true}, p_7 = 0 \wedge p_{12} \in \{\text{off}, \text{on}, \text{collect}\} \wedge p_{14} = g, p_{12} \in \{\text{off}, \text{on}\} \rangle$.

BF(End) =



End

The system is in this mode when no more reactions will be performed. While the system resides in this mode the vessel is empty and RLight is green. The system remains in this mode until it is shutdown.

End = $\langle p_{12} = \text{off}, p_7 = 0 \wedge p_{14} = g, \Omega \rangle$.

Behaviour Function Connection

MRS(CP) is produced by applying function SEM to MPS(CP) and BF (see figure B.4).

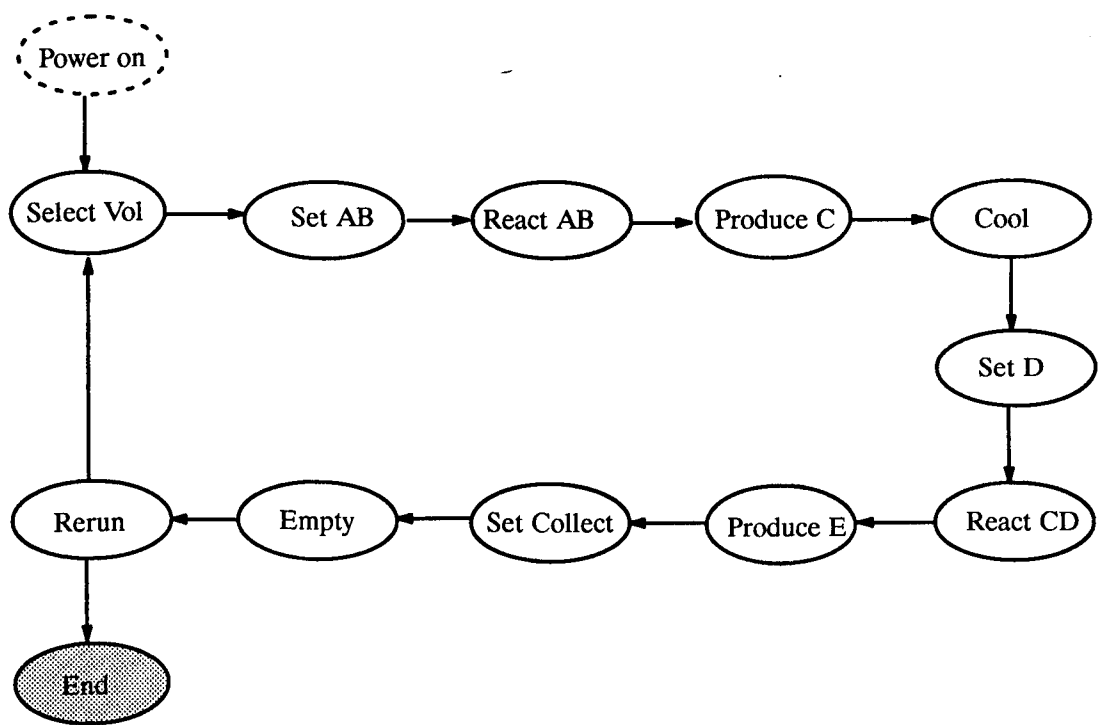


Figure B.4. Mission Real World Specification of Chemical Plant

Mission Real World Description Analysis

In this section we present an overview of how MRD(CP) is produced, by following the guidelines of section 7.8.1.

The mission real world variables are: $p_1, \dots, p_{12}, p_{14}, p_{16}, \dots, p_{22}$. Let us suppose that the analysis of the mission real world variables and mission real world specification, determines that all the relation of SRD(CP) are also relevant for MRD(CP). In addition six new description relations are introduced, these are summarised in table B.8.

Table B.8: Additional Relationships of Mission Real World Description

No.	Related variables	Relationship	Comments
Ir ₅	p_1, p_{12}, p_{14}	$p_1 = S(T) \Rightarrow p_{12} = \text{off} \wedge p_{14} = g.$	At the start of the system lifetime Plant select is at off and RLight at green.
Hr ₈	p_9, \dots, p_{12}	$\forall t: p_{12} \in \{\text{start, collect}\} \Rightarrow$ $\forall t: p_9 = p_{9,0} \wedge p_{10} = p_{10,0} \wedge p_{11} = p_{11,0}$	The set point selectors must remain constant while Plant select is at start or collect.

Hr ₉	p ₂ , p ₄ , p ₈ , p ₁₆ , p ₂₂ .	$p_{4,0}=0 \wedge \forall t: p_8(t) < Y \wedge p_{22}(t)=0 \Rightarrow$ $p_{2,1} = \text{Min}(\int p_{16}(t) dt + p_{2,0}, V_{\text{max}})$	Provided the temperature remains below Y and the outflow rate is zero during any interval, and at the start of the interval there is no C in the vessel, then VolA is equal to the smaller of the sum of the integral of the flow rate over the interval and the maximum level.
Hr ₁₀	p ₃ , p ₄ , p ₈ , p ₁₇ , p ₂₂	$p_{4,0}=0 \wedge \forall t: p_8(t) < Y \wedge p_{22}(t)=0 \Rightarrow$ $p_{3,1} = \text{Min}(\int p_{17}(t) dt + p_{3,0}, V_{\text{max}})$	This relation is the VolB equivalent of Hr ₉ .
Hr ₁₁	p ₅ , p ₆ , p ₈ , p ₁₈ , p ₂₂	$p_{6,0}=0 \wedge \forall t: p_8(t) < Z \wedge p_{22}(t)=0 \Rightarrow$ $p_{5,1} = \text{Min}(\int p_{18} dt + p_{5,0}, V_{\text{max}})$	This relation is the VolD equivalent of Hr ₉ .
HR ₁₂	p ₁₃ , p ₁₈	$T_0 = S(T) \Rightarrow p_{7,0} = 0 \wedge p_{12,0} = \text{off} \wedge$ $p_{14,0} = g.$	At the start of the system lifetime the vessel is empty, Plant select is at off and RLight is at green.

The mission real world description of the chemical plant is given as:

$$\text{MRD}(\text{CP}) = \langle T, \langle p_1, \dots, p_{24} \rangle, \langle V_{p_1}, \dots, V_{p_{24}} \rangle, \langle C_{p_1}, \dots, C_{p_{24}} \rangle, \langle I_{r_1}, \dots, I_{r_5} \rangle, \langle H_{r_1}, \dots, H_{r_{12}} \rangle \rangle.$$

Mode Graph Verification

In this section, an examples of a completeness check and consistency check are presented.

Completeness Check

The completeness of MRS(CP) is checked by confirmation of the following condition.

$$\text{MRS}(\text{CP}) \text{ cmp } \text{MRD}(\text{CP}) \text{ iff}$$

$$\forall x \in M(\text{MRS}(\text{CP})): \forall H \in \text{Set}(\text{MRD}(\text{CP})): \forall \text{Int} \in \text{SI}(T):$$

$$[H \text{ sat } x@ \text{Int} \Rightarrow \exists y \in \text{MRS}(\text{CP}).\text{sr}(x): H \text{ sat } (\text{Start}(y) \wedge \text{Inv}(y) \vee \Omega)@e(\text{Int})].$$

The completeness of the rerun mode is checked below.

We have: Rerun = $\langle \text{true}, p_7 = 0 \wedge p_{12} \in \{\text{off}, \text{on}, \text{collect}\} \wedge p_{14} = g, p_{12} \in \{\text{off}, \text{on}\} \rangle$; and

$$\text{MRS}(\text{CP}).\text{sr}(\text{Rerun}) = \{\text{Select Vol}, \text{End}\}.$$

$$\text{End} = \langle p_{12} = \text{off}, p_7 = 0 \wedge p_{14} = g, \Omega \rangle.$$

$$\text{Select vol} = \langle p_{12} = \text{on}, p_7 = 0 \wedge p_{12} \in \{\text{on}, \text{start}\} \wedge p_{14} = g, p_{12} = \text{start} \rangle.$$

$$\forall H \in \text{Set}(\text{MRD}(\text{CP})): \forall \text{Int} \in \text{SI}(T):$$

$$[H \text{ sat } \text{Rerun}@ \text{Int} \Rightarrow H \text{ sat } p_7 = 0 \wedge p_{14} = g \wedge p_{12} \in \{\text{off}, \text{on}\}@e(\text{Int})].$$

$$\therefore \forall H \in \text{Set}(\text{MRD}(\text{CP})): \forall \text{Int} \in \text{SI}(T): H \text{ sat } \text{Rerun}@ \text{Int} \Rightarrow$$

$$H \text{ sat } (\text{Start}(\text{Select vol}) \wedge \text{Inv}(\text{Select vol})) \vee (\text{Start}(\text{Close}) \wedge \text{Inv}(\text{Close}))@e(\text{Int}).$$

Consistency Checks

The consistency of MRS(CP) is checked by confirmation of the following three conditions.

- i) $\forall x \in S(\text{MRS}(\text{CP})): \exists V \in \Gamma: [V \text{ sat } \text{Start}(x) \wedge \text{Inv}(x) \wedge \neg \text{End}(x) \wedge C(\text{IR}(\text{MRD}(\text{CP})))].$
- ii) $\forall (x, y) \in A(\text{MRS}(\text{CP})): \exists V \in \Gamma: [V \text{ sat } \neg \text{Inv}(x) \wedge \text{End}(x) \wedge \text{Start}(y) \wedge \text{Inv}(y) \wedge \neg \text{End}(y) \wedge C(\text{IR}(\text{MRD}(\text{CP})))].$
- iii) $\forall x \in E(\text{MRS}): \exists V \in \Gamma: [V_x \text{ sat } \text{Inv}(x) \wedge \text{End}(x) \wedge C(\text{IR}(\text{MRD}))].$

In this study, we sketch the proofs for the confirmation of condition i) and condition ii).

Condition i

$S(\text{MRS}) = \text{Power on}, \text{Power on} = \langle p_{12} = \text{off}, p_7 = 0 \wedge p_{12} \in \{\text{off}, \text{on}\} \wedge p_{14} = g, p_{12} = \text{on} \rangle.$

We must show: $\exists V \in \Gamma: [V \text{ sat } p_{12} = \text{off} \wedge p_7 = 0 \wedge p_{14} = g \wedge C(\text{IR}(\text{MRD}))].$

The above assertion, holds since no constraints are imposed over p_7 and p_{14} by the invariant relations., and the invariant relation over p_{12} only restricts its value only at the start point.

Condition ii

A sketch of the proof for the arc (Produce C, Cool) is presented next.

We must show:

$\exists V \in \Gamma:$

$[V \text{ sat } RY \wedge p_{12} = \text{start} \wedge p_{14} = a \wedge ZDE \wedge p_8 \leq Y + \Delta R \wedge ZF \wedge p_8 \geq Z \wedge C(\text{IR}(\text{MRD}))].$

The above assertion, holds since $RY \Rightarrow p_8 \leq Y + \Delta R$, and $Y + \Delta R \geq Z$ no invariant relation is imposed over the variables, involved in this assertion.

Safety Verification of Mission

For the safety verification of the mission it must be verified that for any history that satisfies the mission specification a hazardous states will not occur.

Formally, we must show: $\forall H \in \text{Set}(\text{MRD}): H \text{ sat } \text{MRS}@Int \Rightarrow H \text{ sat } \text{SRS}@Int.$

The safety verification condition for the mission can be restated in terms of the modes as:

$\forall m \in M(\text{MRS}): \forall H \in \text{Set}(\text{MRD}): \forall Int \in SI(T): [H \text{ sat } m@Int \Rightarrow H \text{ sat } \text{SRS}@Int].$

Next an illustrative portion of the safety verification of the mission is presented. This condition is proven for the Produce C mode, the proofs for the other modes can be

constructed in a similar way.

We must prove: $\forall H \in \text{Set}(\text{MRD}): \forall \text{Int} \in \text{SI}(\text{T}): [H \text{ sat Produce C@Int} \Rightarrow H \text{ sat SRS@Int}]$.

We have: $\text{SRS} = (p_6 \leq \text{Tvol} \vee p_8 \leq \text{Tact}) \wedge (p_7 = 0 \vee p_8 \leq \text{Eact})$.

Produce C =

$\langle \text{true}, \text{ZDE} \wedge \text{RY} \wedge p_{12} = \text{start} \wedge p_{14} \in \{a, r\} \wedge \text{ZF}, p_{14} = a, \text{CL}(p_9, p_{10}), \text{CL}(p_9, p_{10}) + \Delta C \rangle$.

$\forall H \in \text{Set}(\text{MRD}): \forall \text{Int} \in \text{SI}(\text{T}): [H \text{ sat Produce C@Int} \Rightarrow H \text{ sat } p_6 = 0 @\text{Int}]$.

This follows from the factor ZDE of the invariant of Produce C.

$\forall H \in \text{Set}(\text{MRD}): \forall \text{Int} \in \text{SI}(\text{T}): [H \text{ sat Produce C@Int} \Rightarrow H \text{ sat } p_8 \leq Y + \Delta R @\text{Int}]$.

This follows from the factor RY of the invariant of Produce C. Hence SRS is satisfied during the Produce C mode, if $Y + \Delta R < \text{Eact}$.

Mission Validation

Let us suppose that the customer confirms that MRS(CP) accurately captures the mission requirements of the system. However, the customer (after consultation with the operator), imposes an additional requirement: “The fact that a batch of E has been collected should be indicated to the operator. This should be achieved by an indicator that is turned on after E has been collected. ”

This requirement change introduces another real world variable and two additional modes. The new variable (p_{25}) represents the state of the indicator for the collection of E (referred to as ELight); this variable has range {on, off} and is in the class Free. The invariants of the modes of MRS(CP) must be modified by the conjunction of the predicate $p_{25} = \text{off}$. The additional modes are the signal on and signal off modes; these are defined below. The structure of the modified mission real world specification is given in figure B.5.

Signal on mode

The system is in this mode while ELight is being set. At the start of this mode ELight is off. While the system resides in this mode the vessel is empty, Plant select is at collect and RLight is green. The system leaves this mode as soon as ELight is on. The system must spend at most SGu seconds in this mode, where SGu is an upper bound on the time required to change the

status of ELight.

Signal on = $\langle p_{25} = \text{off}, p_7 = 0 \wedge p_{12} = \text{collect} \wedge p_{14} = g, p_{25} = \text{on}, 0, \text{SGu} \rangle$.

Signal off mode

The system is in this mode while ELight is being reset. At the start of this mode ELight is on. While the system resides in this mode the vessel is empty, Plant select is at on and RLight is green. The system leaves this mode as soon as ELight is off. The system must spend at most SGu seconds in this mode.

Signal off = $\langle p_{25} = \text{on}, p_7 = 0 \wedge p_{12} = \text{on} \wedge p_{14} = g, p_{25} = \text{off}, 0, \text{SGu} \rangle$.

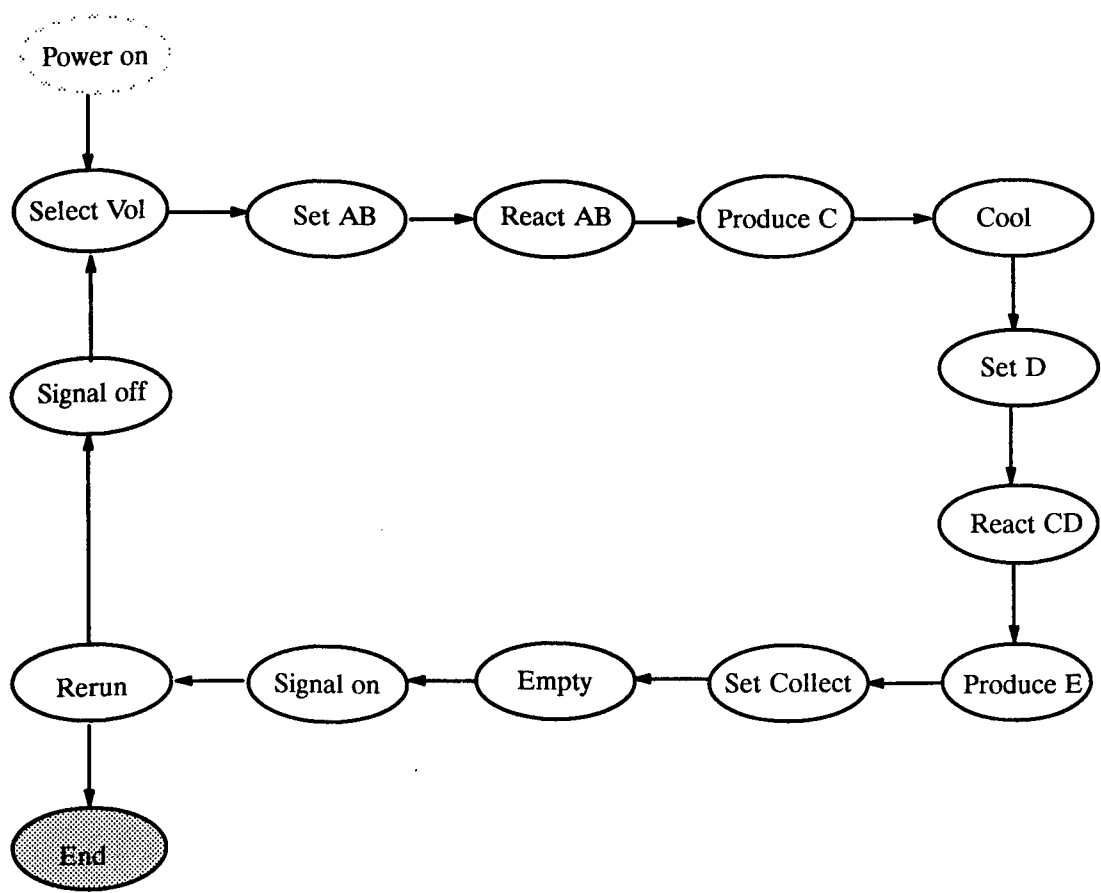


Figure B.5. Validated Mission Real World Specification of Chemical Plant

B.6 Summary

As a result of the analysis presented in this appendix, the four real world specification have been produced. The safety real world analysis of the chemical plant resulted in the formulation of the hazards as HS(CP), the safety real world requirements as SRS(CP); and the restrictions imposed on the safety-critical behavior as SRD(CP). The mission real world analysis of the chemical plant resulted in the formulation of the mission real world requirements as MRS(CP) and the restriction imposed on mission-oriented behaviour as MRD(CP). In appendix C, the derivation of the controller specifications from these real world specifications is illustrated.

Appendix C - Controller Analysis

C.1. Introduction

In this appendix the controller analysis of the safety-critical chemical plant, introduced in appendix B, is described. This case study is presented to illustrate how the guidelines of chapter 8 provide a systematic approach to the controller analysis of a system. The specifications produced at each stage of the analysis are presented. To keep the example manageable, only illustrative portions of the analysis and proof are discussed. In particular, the case study concentrates on the proofs which are necessary to confirm that the safety controller specification is adequate for the safety real world specification.

At the start of the controller analysis, the safety-critical behaviour of the chemical plant is expressed formally by SRD(CP) and SRS(CP); and the mission-oriented behaviour by MRD(CP) and MRS(CP). These formal constructs were developed by performing a real world analysis of the chemical plant (see appendix B).

C.2. Safety Controller Analysis

In this section, I will describe the safety controller analysis of the chemical plant. This analysis will lead to the production of SED(CP) and SCS(CP).

C.2.1. Safety Environment Description

The analysis performed to produce SED(CP) follows the guidelines given in section 8.2.

Step 1

Let us suppose that the actuators of the safety controller consist of a lock on each inlet and a valve for OutletS, and the sensors are a thermometer and a volume sensor. The variables that model the state of these actuators and sensors are given in table C.1.

Table C.1: Safety Controller Variables

Notation	Units	Name	Comments
p26	Lk_pos	LockA	The position of the lock for InletA.
p27	Lk_pos	LockB	The position of the lock for InletB.
p28	Lk_pos	LockD	The position of the lock for InletD.
p29	mm	SValve	The extent to which the valve for OutletS is open.

p30	°K	Thermometer	The thermometer reading for the contents of the vessel.
p31	dm ³	VolSensor	The volume sensor reading for the vessel.

Step 2

A systematic analysis to identify the ranges and classes of the variables identified in *step 1* is performed. The results of this analysis are presented in the following two tables.

Table C.2: Ranges for Safety Controller Variables

Variables	Range	Comments
p26, p27, p28	{off, mid, on}	Where off, mid and on are the states of the locks.
p29	$\{x \in \mathbb{R} \mid 0 \leq x \leq S_{\max}\}$	Where S_{\max} is the maximum width to which SValve can be opened.
p30	$\{T_l, T_l + \Delta q, \dots, T_l + Q \cdot \Delta q\}$	Where T_l is the lower limit of the thermometer, Δq the granularity and $T_l + Q \cdot \Delta q$ the upper limit.
p31	$\{0, \Delta v, \dots, V \cdot \Delta v\}$	Where Δv is the granularity and $V \cdot \Delta v$ the upper limit of the volume sensor.

Table C.3: Categories and Classes for Safety Controller Variables

Variables	Category	Class	Comments
p26, p27, p28, p30, p31	Controller	Free	An analysis of the locks and the thermometer shows that they are in the class free.
p29	Controller	Continuous	SValve is a continuous variable.

Step 3

The basic relationships involving a sensor (or actuator) are formulated by inspection of its specification. For example, let us suppose that the specification of the lock for InletA states that no chemical can flow through InletA when the lock is on, hence the relation Ir_6 is added to the description relations table.

As another example, let us suppose that an analysis of the state of the system at the start of the system lifetime shows that SafeLight is green, and the locks are all on this leads to relation Ir_{12} . The basic relationships of the actuators and sensors are summarised in table C.4.

Table C.4: Controller Relationships for Safety Environment Description

No.	Related variables	Relationship	Comments
Ir ₆	p ₁₆ , p ₂₆	$p_{26} = \text{on} \Rightarrow p_{16} = 0$	If LockA is on, no A flows into the vessel.
Ir ₇	p ₁₇ , p ₂₇	$p_{27} = \text{on} \Rightarrow p_{17} = 0$	If LockB is on, no B flows into the vessel.
Ir ₈	p ₁₈ , p ₂₈	$p_{28} = \text{on} \Rightarrow p_{18} = 0$	If LockD is on, no D flows into the vessel.
Ir ₉	p ₇ , p ₂₁ , p ₂₉	$p_{21} = F_{OS}(p_7, p_{29})$	OutflowS is the result of applying F _{OS} to Vol and Svalve.
Ir ₁₀	p ₈ , p ₃₀	$ p_8 - p_{30} \leq \Delta T_p$	The imprecision of the Thermometer is bounded by ΔT_p .
Ir ₁₁	p ₇ , p ₃₁	$ p_7 - p_{31} \leq \Delta V_p$	The imprecision of the volume sensor is bounded by ΔV_p .
Ir ₁₂	p ₁ , p ₁₅ , p ₂₆ , p ₂₇ , p ₂₈	$p_1 = s(T) \Rightarrow$ $p_{15} = g \wedge p_{26} = \text{on} \wedge p_{27} = \text{on} \wedge$ $p_{28} = \text{on}.$	At the start of the system lifetime SafeLight is at green and the locks are all on.

Step 4

The safety environment description of the chemical plant is given below:

$SED(CP) = \langle T, \langle p_1, \dots, p_{31} \rangle, \langle Cp_1, \dots, Cp_{31} \rangle, \langle Ir_1, \dots, Ir_4, Ir_6, \dots, Ir_{12} \rangle, \langle Hr_1, \dots, Hr_7 \rangle \rangle.$

The interactions between the components of the safety controller, the safety operator and plant are illustrated in figure C.1.

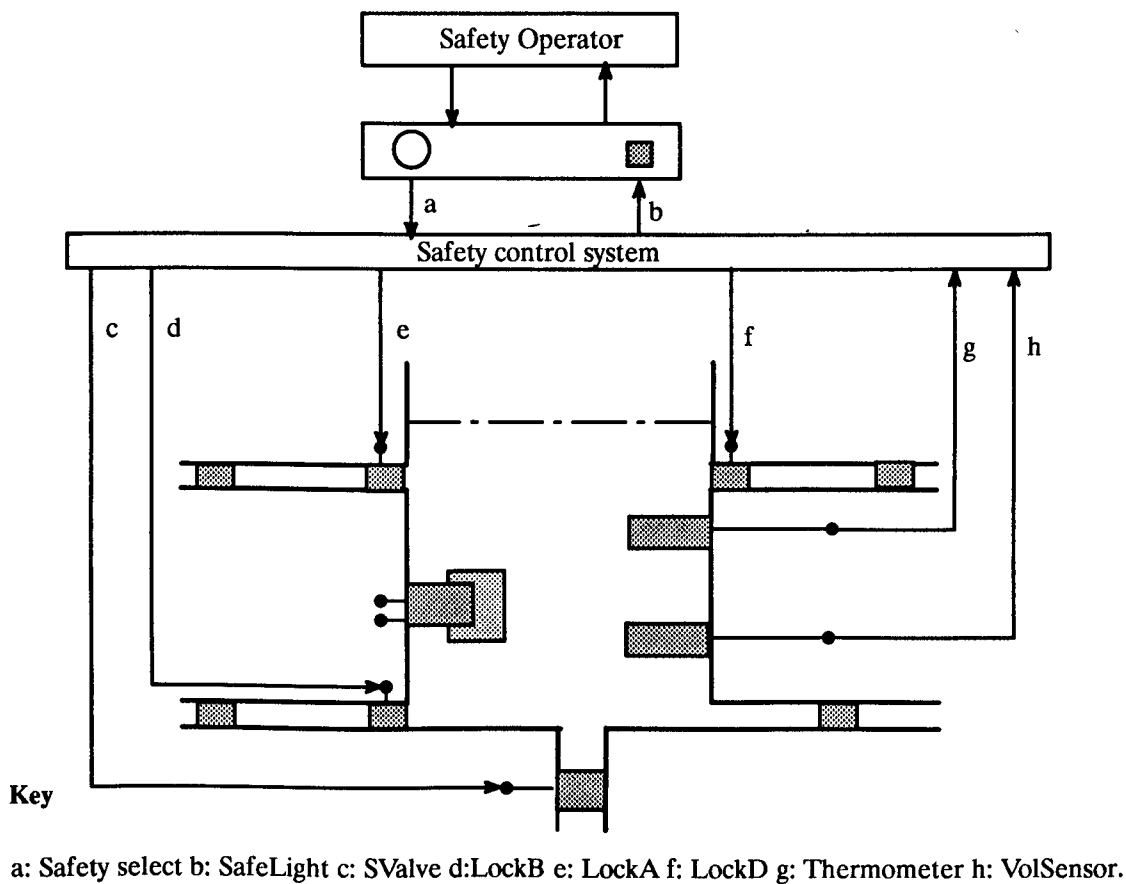


Figure C.1. Chemical Plant Safety Controller

C.1.2. Safety Controller Specification Analysis

Here the production of the safety controller specification, by following the guidelines of section 8.3.2, is discussed.

The safety real world specification of the chemical plant is recalled:

$$SRS(CP) = (p_6 < Tvol \vee p_8 \leq Tact) \wedge (p_7 = 0 \vee p_8 \leq Eact).$$

Also recall that the relationship between the colour of SafeLight and the status of the reaction vessel should obey the following (informal) rules:

- it should be *green* when a reaction is not in progress;
- it should be *amber* when a reaction may be in progress; and
- it should be *red* when the safety controller must override the mission controller to prevent the system from entering into a hazardous state.

Mode Graph Production

Here, the mode graphs are produced for the six phases of the general safety controller structure (i.e. Start up, Monitor, Recovery, Reset, Shut down and End phases) by following the guidelines for the phases. This case study concentrates on the production rules derived from the safety verification checks; these rules are recalled below. (In the following the clause numbers refer to the safety verification check.)

The rules *a* and *b* are defined below for a mode graph *MG* and precondition function *PF*; rule *a* is used to confirm clause *ii* for *MG* and rule *b* to confirm clause *iii* for *MG*.

Rule a. For any arc (x, y) of $A(MG)$, any history H of $SEDH$ and any interval Int , if H satisfies the precondition of x at $s(Int)$ and x during Int and the start and invariant predicates of y at $e(Int)$ then H satisfies the precondition of y at $e(Int)$.

$\forall (x, y) \in A(MG): \forall H \in SEDH: \forall Int \in SI(T):$

$H \text{ sat } PF(x)@s(Int) \wedge H \text{ sat } x@Int \wedge H \text{ sat } (Start(y) \wedge Inv(y))@e(Int) \Rightarrow$

$H \text{ sat } PF(y)@e(Int).$

Rule b. For any mode m of MG , any history H of $SEDH$ and any interval Int , if H satisfies the precondition of m at $s(Int)$ and m during Int then H must satisfy *SRS* during Int .

$\forall m \in M(MG): \forall H \in SEDH: \forall Int \in SI(T):$

$H \text{ sat } PF(m)@s(Int) \wedge H \text{ sat } m@Int \Rightarrow H \text{ sat } SRS@Int.$

The rule *c* is defined below for a pair of modes (x, y) and precondition function *PF*; rule *c* is used to confirm clause *ii* for (x, y) .

Rule c. For any history H of $SEDH$ and any interval Int , if H satisfies the precondition of x at $s(Int)$ and x during Int and the start and invariant predicates of y at $e(Int)$ then H satisfies the precondition of y at $e(Int)$.

$\forall H \in SEDH: \forall Int \in SI(T):$

$H \text{ sat } PF(x)@s(Int) \wedge H \text{ sat } x@Int \wedge H \text{ sat } (Start(y) \wedge Inv(y))@e(Int) \Rightarrow H \text{ sat } PF(y)@e(Int).$

In the proofs of the production rules, H will be used to denote an arbitrary history from $SEDH(CP)$ and Int an arbitrary interval from $SI(T)$. In the construction of these proofs, it is shown how the *time bound constraints* for the bounded modes are related to real world

behaviour. The proofs also allow the safety controller analysts to identify any assumptions which cannot be derived from the relations of SED(CP).

Initial Predicate

The initial predicate follows directly from Ir_4 and Ir_{12} .

$$IP = p_7 = 0 \wedge p_8 < \text{Stemp} \wedge p_{15} = g \wedge p_{26} = \text{on} \wedge p_{27} = \text{on} \wedge p_{28} = \text{on}.$$

Step 1 (Start up Phase).

a. Firstly, we define a basic strategy for the Start up phase. This is simply, that the safety controller must wait until Safety select is on and then release the locks on InletA and InletB. This strategy can be specified by two simple tasks: *Safe on* and *Release*.

b. These simple tasks can be specified by modes, which would result in the mode graph below (figure C.2). During the *Safe on* task the safety controller must keep the locks on, and the task is completed when Plant select is set to on. During the *Release* task the safety controller must release the locks on InletA and InletB, within a small duration $U1$.

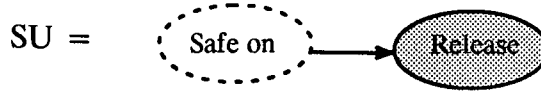


Figure. C.2. Start up Graph

c. In this step, we define the precondition function PF for *Safe on* and *Release*.

i. In this step we investigate the preconditions which are necessary for a mode to pass production rule b.

At the start of *Safe on*, the vessel is empty (from IP) and during *Safe on* the locks are on, therefore the vessel is empty during *Safe on*. Hence *Safe on* would pass rule b. From Hr_5 , the maximum rise in temperature during *Release* is $\Delta Tm.U1$. Hence *Release* will pass rule b, if $PF(\text{Release}) \Rightarrow p_8 < Eact - \Delta Tm.U1$. We define $PC(\text{Release}) = p_8 < Eact - \Delta Tm.U1$.

ii. In this step, we consider if the preconditions identified in the above step will pass rule a. This check is performed by the investigation of the behaviour of p_8 during the predecessors of *Release*. *Release* has only one predecessor *Safe on*. At the end of *Safe on* no constraint is imposed on p_8 . Hence a suitable $PF(\text{Release})$ will not pass rule a for the arc (*Safe on*,

Release).

An obvious solution is for the safety controller to remain in *Safe on*, until Safety select is at on and the Thermometer reading is below a safe value $Stemp - \Delta Tp$. In this case, it would follow (from Ir_{10}) that the temperature is below $Stemp$ at the start of *Release*.

By inspecting the behaviour of the physical process during the *Safe on* mode we define $PF(Release) = p_7 = 0 \wedge p_8 < Stemp$. Hence *Release* will pass rule *b*, if $Stemp < Eact - \Delta Tm.U1$

d. As a result of the analysis performed above, the mode specifications, for the *Safe on* and *Release* modes are given below. These mode specifications also include the necessary conditions over *SafeLight*.

Safe on mode

The safety controller is in this mode while it is waiting to be switched to the monitor phase. During this mode, SafeLight must be at green and the locks on. The safety controller must leave the mode as soon as Safety Select is on and Thermometer is less than $Stemp - \Delta Tp$.

$Safe\ on = \langle true, p_{15} = g \wedge LRset, p_{13} = on \wedge p_{30} < Stemp - \Delta Tp \rangle$,

where $LRset = (p_{26} = on \wedge p_{27} = on \wedge p_{28} = on)$.

Release mode

The safety controller is in this mode while LockA and LockB are being released (to allow chemicals A and B to be loaded into the vessel) and SValve is being closed. At the start of this mode Safety select will be at on. During this mode, SafeLight must be at green or amber and LockD on. The safety controller must set SafeLight to amber, release LockA and LockB, close SValve and then must leave the mode. The safety controller must spend at most U1 seconds in the mode.

$Release = \langle p_{13} = on, p_{15} \in \{g, a\} \wedge p_{28} = on, p_{15} = a \wedge Vset, 0, U1 \rangle$,

where $Vset = (p_{26} = off \wedge p_{27} = off \wedge p_{29} = 0)$.

e. In this step we should confirm that SU is consistent and complete. For this example, we will simply assume that this is the case.

f. Proofs for production rule *a*, for the arc (*Safe on*, *Release*), and rule *b* for the *Safe on* and *Release* modes are given below.

Safe on, Release arc

For rule *a* we show:
$$\begin{aligned} & H \text{ sat } p_7=0 \wedge p_8 < \text{Stemp}@s(\text{Int}) \wedge H \text{ sat } \text{Safe on}@(Int) \\ & \Rightarrow H \text{ sat } p_7=0 \wedge p_8 < \text{Stemp}@e(Int). \end{aligned}$$

Proof. Since the vessel is empty ($p_7=0$) at $s(Int)$ and from the invariant of *Safe on* the locks are on during *Int* then, from relations Ir_2 , Ir_6 , Ir_7 , Ir_8 and Hr_1 , $H \text{ sat } p_7=0@Int$. From the end predicate the thermometer reading is less than $\text{Stemp}-\Delta T_p$ at $e(Int)$, hence from Ir_{10} we have $H \text{ sat } p_8 < \text{Stemp}@e(Int)$

Safe on mode

For rule *b* we show: $H \text{ sat } p_7=0@s(Int) \wedge H \text{ sat } \text{Safe on}@(Int) \Rightarrow H \text{ sat } p_7=0@Int$.

Proof. Follows from proof of rule *a* (see above).

Release mode.

For rule *b* we show:
$$\begin{aligned} & H \text{ sat } (p_7=0 \wedge p_8 \leq \text{Stemp})@s(Int) \wedge H \text{ sat } \text{Release}@(Int) \\ & \Rightarrow H \text{ sat } (p_6=0 \wedge p_8 < \text{Eact})@Int. \end{aligned}$$

Proof. Since the vessel is empty ($p_7=0$) at $s(Int)$ and from the invariant of *Release*, *LockD* is on ($p_{28}=\text{on}$) during *Int* then, from relations Ir_1 , Ir_8 , and Hr_7 , $H \text{ sat } p_6=0@Int$. From the upper bound of *Release* and relation Hr_5 the maximum rise in the temperature from $s(Int)$ to $e(Int)$ is bounded by $U1.\Delta T_m$. Hence, since p_8 is a continuous variable, $H \text{ sat } p_8 < \text{Eact}@Int$, if $U1 < (\text{Eact} - \text{Stemp})/\Delta T_m$.

Step 2 (Monitor Phase)

a. In this step, we construct a phase graph that specifies the behaviour of the physical process as measured by the variables of the safety real world specification.

A phase graph of the safety-critical behaviour of the physical process during a mission of the chemical plant is shown in figure C.3, and the behaviour during the phases is discussed below.

Load AB. The required volumes of A and B are loaded into the vessel; the end of this phase is marked by the temperature rising to at least *Y*.

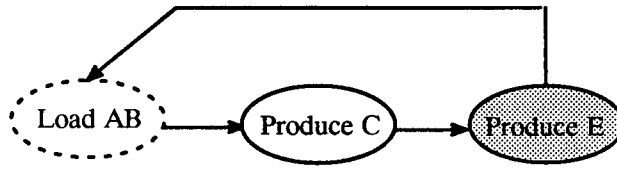


Figure C.3. Safety-Critical View of Mission

Produce C. The reaction to produce C is in progress; the end of this phase is marked by the temperature dropping below Z.

Produce E. The required volume of D is loaded into the vessel; the reaction to produce E is completed and the product collected; the end of this phase is marked by an empty vessel.

b. In this step, we construct a phase graph over the safety controller that monitors the behaviour of the graph identified above.

A phase graph over the sensors of the safety controller that monitor the mission as it passes through the phases defined in *step 2.a*, is shown in figure C.4.

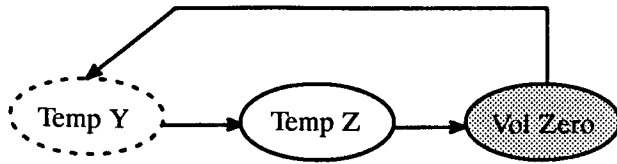


Figure C.4. Safety-Controller View of Mission

The behaviour monitored by the phases of the above graph and the conditions which hold at the end of these phases are discussed next.

Temp Y. This phase monitors the behaviour characterized by the *Load AB* phase. At the end of this phase the thermometer is at least $Y - \Delta Tp$.

Temp Z. This phase monitors the behaviour characterized by the *Produce C* phase. At the end of this phase the thermometer is below $Z - \Delta Tp$.

Vol Zero. This phase monitors the behaviour characterized by the *Produce E* phase. At the end of this phase VolSensor is at most ΔVp .

c. The monitor graph MN is constructed by considering how the safety controller can maintain the *SRS* in a phase of the phase graph of *step 2.b*, while allowing the mission controller to achieve the task monitored by that phase.

Temp Y

- i. The basic strategy for this phase is for the safety controller to hold the actuators constant and leave this phase when the thermometer reading is at least $Y - \Delta T_p$.
- ii. The above strategy can be defined by the mode: *No React*. Roughly speaking, during *No React* the safety controller must keep LockA and LockB off, LockD on and SValve Closed; the safety controller must leave this mode as soon as the thermometer reading is at least $Y - \Delta T_p$.
- iii. In this step we define $PF(\text{No React})$.

Firstly we identify the condition necessary for *No React* to pass production rule *b*. The safety controller must leave this mode as soon as the thermometer reading is at least $Y - \Delta T_p$. Hence, if the temperature is below Y at the start of *No React*, from relation Ir_{10} the temperature will be below Y (and so, below E_{act}) during this mode. During this mode no D enters the vessel, hence from relation Hr_7 provided $p_6 + 2.p_5$ is less than T_{vol} at the start of this mode, $p_6 < T_{vol}$ during this mode. Hence this mode will pass rule *b* if $PF(\text{No React}) \Rightarrow p_8 < Y \wedge p_6 + 2.p_5 < T_{vol}$. We define $PC(\text{No React}) = p_8 < Y \wedge p_6 + 2.p_5 < T_{vol}$

Secondly, we check if the precondition defined in the step above, will pass rule *a* or *c*. This check is performed by investigating the behaviour of p_5 , p_6 and p_8 during the predecessor of *No React* (since *No React* will be the start mode of MN it must pass rule *c* for the arc (*Release*, *No React*). At the end of *Release* we have $p_5 = 0 \wedge p_6 = 0 \wedge p_8 < Stemp + U1.\Delta T_m$. Hence, we conclude that $PF(\text{No React})$ will hold at the end of *Release*, if $Stemp + U1.\Delta T_m < Y$.

We define $PF(\text{No React}) = p_8 < Y \wedge p_6 + 2.p_5 < T_{vol} - \Delta V$, where ΔV is a small tolerance.

- iv. In this step, we construct the mode specification of *No React*; this specification also includes the necessary condition over SafeLight.

No React mode

The safety controller is in this mode while the vessel is being loaded with the volumes of A and B. During this mode, SafeLight is at amber, LockA and LockB are off, SValve is closed and LockD is on. The safety controller must leave this mode as soon as Thermometer is at least $Y -$

ΔTp

No React = $\langle \text{true}, p_{15} = a \wedge V_{\text{set}} \wedge p_{28} = \text{on}, p_{30} \geq Y - \Delta Tp \rangle$.

Temp Z

By following the four steps for the construction of a mode graph and definition of precondition for the Temp Z phase (as was done for Temp Y phase) we conclude that the behaviour of the safety controller can be defined by the *React C* mode, and identify the precondition given below.

$\text{PF}(\text{React C}) = p_6 + 2.p_5 < T_{\text{vol}} - \Delta V \wedge p_8 \leq Y$;

React C mode

The safety controller is in this mode while a reaction for the production of C may be in progress. During this mode, Safelight is at amber, LockA and LockB are off, SValve is closed and LockD is on. The safety controller must leave this mode as soon as Thermometer is at least $E_{\text{act}} - (\Delta Tp + \Delta RT)$ or less than $Z - \Delta Tp$.

$\text{React C} = \langle \text{true}, p_{15} = a \wedge V_{\text{set}} \wedge p_{28} = \text{on}, p_{30} \geq E_{\text{act}} - (\Delta Tp + \Delta RT) \vee p_{30} < Z - \Delta Tp \rangle$.

Vol Zero

By following the four steps for the construction of a mode graph and definition of precondition for the Vol Zero phase (as was done for Temp Y phase) we conclude that the behaviour of the safety controller can be defined by the mode graph given in figure C.5, and identify the preconditions given below.



Figure C.5. Mode graph for Vol Zero

$\text{PF}(\text{Release D}) = p_8 \leq Z$;

$\text{PF}(\text{React E}) = p_8 \leq Z + U2.\Delta T_m$;

and $\text{PF}(\text{Lock}) = p_7 \leq 2.\Delta V_p \wedge p_8 < S_{\text{temp}}$.

Release D mode

The safety controller is in this mode while the Lock on Inlet D is being released. At the start of this mode the Thermometer is less than $Z - \Delta T_p$. During this mode, SafeLight is at amber, LockA and LockB are off and SValve is closed. The safety controller must release LockD and then leave this mode and can spend at most U2 seconds in this mode.

React E = $\langle p_{30} < Z - \Delta T_p, p_{15} = a \wedge V_{set}, p_{28} = \text{off}, 0, U_2 \rangle$.

React E mode

The safety controller is in this mode while a reaction for the production of E may be in progress. During this mode, Safelight is at amber, the locks are off and SValve is closed. The safety controller must leave this mode as soon as Thermometer is at least $T_{act} - (\Delta T_p + \Delta RT)$ or less than $Stemp - \Delta T_p$ and VolSensor at most ΔV_p .

React E =

$\langle \text{true}, p_{15} = a \wedge V_{set} \wedge p_{28} = \text{off}, p_{30} \geq T_{act} - (\Delta T_p + \Delta RT) \vee (p_{30} < Stemp - \Delta T_p \wedge p_{31} \leq \Delta V_p) \rangle$.

Lock mode

The safety controller is in this mode while LockD is being locked. At the start of this mode, Thermometer is less than $Stemp - \Delta T_p$ and VolSensor at most ΔV_p . During this mode, Safelight is at amber, LockA and LockB are off and SValve is closed. The safety controller must turn LockD on, then leave this mode. The safety controller must spend at most U3 seconds in this mode.

Lock = $\langle p_{30} < Stemp - \Delta T_p \wedge p_{31} \leq \Delta V_p, p_{15} = a \wedge V_{set}, p_{28} = \text{on}, 0, U_3 \rangle$

d. The monitor graph MN for the chemical plant is obtained, by combining the mode graph constructed for the three phases Temp Y, Temp Z and Vol Zero. MN is given in the figure C.6.



Figure C.6. Monitor graph

e. In this step we must check the consistency of MN. Let us suppose that by performing the consistency checks it has been shown that MN is consistent.

f. In this step, we must confirm that production rules *a* and *b* hold for the monitor graph. Proofs of production rule *a* are given for the arc (*No React*, *React C*) and (*Release D*, *React E*) and (*Lock*, *No React*), a proof of rule *b* is given for *Lock*.

No React, React C Arc

For rule *a* we show:

$$\begin{aligned} & H \text{ sat } (p_6 + 2.p_5 < Tvol - \Delta V \wedge p_8 < Y) @ s(Int) \wedge H \text{ sat } No \text{ React} @ (Int) \\ \Rightarrow & H \text{ sat } (p_6 + 2.p_5 < Tvol - \Delta V \wedge p_8 \leq Y) @ e(Int). \end{aligned}$$

Proof. Since $p_6 + 2.p_5 < Tvol - \Delta V$ at $s(Int)$ and from the invariant of *No react* Lock D is on ($p_{28} = on$) during *Int*, then from Ir₈ and Hr₇: $H \text{ sat } p_6 + 2.p_5 < Tvol - \Delta V @ e(Int)$. The end predicate of *No react* is $p_{30} \geq Y - \Delta Tp$, hence $H \text{ sat } p_{30} < Y - \Delta Tp @ Int - \{e(Int)\}$, hence, from Ir₁₀, $H \text{ sat } p_8 < Y @ Int - \{e(Int)\}$. If $dur(Int) > 0$, $H \text{ sat } p_8 \leq Y @ e(Int)$ since p_8 is a continuous variable and if $dur(Int) = 0$ then $H \text{ sat } p_8 \leq Y @ e(Int)$ since $e(Int) = s(Int)$.

Release D, React E Arc

$$\begin{aligned} \text{For rule } a \text{ we show: } & H \text{ sat } p_8 \leq Z @ s(Int) \wedge H \text{ sat } Release \ D @ (Int) \\ \Rightarrow & H \text{ sat } p_8 \leq Z + U2.\Delta Tm @ e(Int). \end{aligned}$$

Proof. From the upper bound of *Release D* and Hr₅ the maximum rise in the temperature from $s(Int)$ to $e(Int)$ is bounded by $U2.\Delta Tm$. Therefore $H \text{ sat } p_8 < Z + U2.\Delta Tm @ e(Int)$.

Lock , No React Arc

$$\begin{aligned} \text{For rule } a \text{ we show: } & H \text{ sat } p_7 \leq 2.\Delta Vp \wedge p_8 < Stemp @ s(Int) \wedge H \text{ sat } Lock @ (Int) \\ \Rightarrow & H \text{ sat } p_6 + 2.p_5 < Tvol - \Delta V \wedge p_8 < Y @ e(Int). \end{aligned}$$

Proof. From the upper bound of *Lock* and Hr₅ the rise in the temperature from $s(Int)$ to $e(Int)$ is bounded by $U3.\Delta Tm$. Hence, $H \text{ sat } p_8 < Y @ e(Int)$, if $U3 < (Y - Stemp) / \Delta Tm$. Similarly, $H \text{ sat } p_6 + 2.p_5 < Tvol - \Delta V @ e(Int)$, if $U3 < (Tvol - \Delta V - 2.\Delta Vp) / FmaxD$.

Lock mode.

$$\begin{aligned} \text{For rule } b \text{ we show: } & H \text{ sat } p_8 < Stemp @ s(Int) \wedge H \text{ sat } Lock @ Int \\ \Rightarrow & H \text{ sat } p_8 < Tact @ e(Int). \end{aligned}$$

Proof. From the upper bound of *Lock* and Hr_2 the rise in the temperature from $s(Int)$ to $e(Int)$ is bounded by $U3 \cdot \Delta Tm$. Hence, $H \text{ sat } p_8 < Tact @ e(Int)$, if $U3 < (Tact - Stemp) / \Delta Tm$. (It should be noted that $p_8 < Tact \Rightarrow SRS$)

The proofs for production rule *a* for the arcs (*React C*, *Release D*) and (*React E*, *Lock*) are outlined below.

React C. The fact that predicate $p_8 \leq Z$ holds at the start of *Release D* follows from the start predicate of *Release D* and relation Ir_8 .

React E. The fact that $p_7 \leq 2 \cdot \Delta Vp \wedge p_8 < Stemp$ holds at the start of the *Lock* mode follows from the start predicate of *Lock* and the relations Ir_{10} and Ir_{11} .

The proofs for production rule *b* for *No React*, *React C*, *Release D* and *React E* are briefly outlined below. For *No React* and *React C* it is argued that the volume of *E* in the vessel is below $Tvol$ and the temperature is below *Eact*. For *Release D* and *React E* it is argued that the temperature is below *Tact*.

No React and *React C.* At the start of these modes $p_6 + 2 \cdot p_5 < Tvol - \Delta V$; since *Lock D* is on during the modes, from relations Ir_8 , and Hr_7 we have $p_6 + 2 \cdot p_5 < Tvol - \Delta V$ during these modes. At the start of *No React*, $p_8 < Y$ and up to the end point $p_8 < Y$. Hence, since $Y < Eact$ and p_8 is a continuous variable, $p_8 < Eact$ during *No React*. At the start of *React C*, $p_8 \leq Y$ and up to the end point $p_8 < Eact - \Delta RT$. Hence, since $\Delta RT > 0$, $Y < Eact$ and p_8 is a continuous variable, $p_8 < Eact$ during *React C*.

Release D and *React E.* From proof for rule *a* of *Release D*, $p_8 \leq Z + U2 \cdot \Delta Tm$ during the mode. Hence $p_8 \leq Tact$, if $U2 < (Tact - Z) / \Delta Tm$. At the start of *React E* $p_8 < Z$. From the end predicate of *React E* and relation Ir_8 we have that $p_8 < Tact - RT$ up to the end of *React E*. Hence, since $RT > 0$, $Z < Tact$ and p_8 is a continuous variable, $p_8 < Eact$ during *React E*.

g. In this step we must confirm production rules *c*, *d* and *e* for the arc (*Release*, *No React*). A proof for rule *c* is given below.

$$\begin{aligned} & H \text{ sat } (p_7 = 0 \wedge p_8 < Stemp) @ s(Int) \wedge H \text{ sat } Release @ (Int) \\ \Rightarrow & H \text{ sat } (p_6 + 2 \cdot p_5 < Tvol - \Delta V \wedge p_8 < Y) @ e(Int). \end{aligned}$$

Proof. Since $p_7 = 0$ at $s(Int)$ and from the invariant of *Release Lock D* is on ($p_{28} = on$) during

Int then, from Ir_8, Hr_7 : $H \text{ sat } p_6 = 0 @ e(Int)$. From the upper bound of *Release* and Hr_5 the rise in the temperature from $s(Int)$ to $e(Int)$ is bounded by $U1 \cdot \Delta T_m$. Hence, $H \text{ sat } p_8 < Y @ e(Int)$, if $U1 < (Y - Stemp) / \Delta T_m$.

Step 3 (Recovery Phase)

a. In this step, we identify all the monitor modes for which a recovery graph is necessary. These modes are recorded in set REM. A recovery graph is not needed for a mode if after the end of the mode the safety controller will remain in the monitor phase. Hence, a mode m of the monitor mode does not need a recovery graph if it satisfies the following condition:

$$\forall H \in SEDH: \forall Int \in SI(T): H \text{ sat } m @ Int \Rightarrow \exists x \in MN.sr(m): H \text{ sat } Start(x) \wedge Inv(x) @ e(Int).$$

Next we must check the modes *No React*, *React C*, *Release D*, *React E* and *Lock* against the above condition.

No React. The successor of this mode is *React C*. The conjunction of the start and invariant of *React C* gives the system predicate: $p_{15} = a \wedge Vset \wedge p_{28} = on$. This system predicate is, in fact, the invariant of *No React*. Therefore *No React* satisfies the above condition. Hence *No React* does not require a recovery graph.

React C. The successor of this mode is *Release D*. The conjunction of the start and invariant of *Release D* gives the system predicate: $p_{30} < Z - \Delta T_p \wedge p_{15} = a \wedge Vset$. However, the only constraint over the thermometer is given by the end predicate of *React C*: $p_{30} \geq Eact - (\Delta T_p + \Delta RT) \vee p_{30} < Z - \Delta T_p$. Clearly, this system predicate does not imply $p_{30} < Z - \Delta T_p$. Therefore *React C* does not satisfy the above condition. Hence *React C* requires a recovery graph and is included in the set REM.

As a result of similar analyses for *Release D*, *React E* and *Lock*, we conclude that *Release D* and *Lock* do not require Recovery graphs, but *React E* does.

Hence, we define REM as: $\{React C, React E\}$.

b. In this step we define the start predicate of the start modes of the recovery graphs.

$$Start(Rec(React C)) = p_{30} \geq Eact - (\Delta T_p + \Delta RT).$$

$$Start(Rec(React E)) = p_{30} \geq Tact - (\Delta T_p + \Delta RT).$$

c. In this step, we define the basic recovery strategies for the modes of REM.

For the sake of simplicity, we only consider the recovery graph for *React C*. The basic recovery strategy is to lock all Inlets and empty the vessel before the system enters a hazardous state. The vessel must then remain empty until the safety operator selects the reset option by setting Safety select to on.

d. In this step we construct the mode graph for REC(React C) and extend PF to cover the modes of this mode graph.

i. We can identify three tasks for the safety controller: *Lock AB*, *Drain*, *Empty*. During the *Lock AB* task the safety controller must lock the Inlets and open SValve, within a duration U4. During the *Drain* task, the safety controller must keep the Inlets locked, and SValve open, long enough to empty the vessel, and then signal the completion of the recovery, within a duration U5. During the *Empty* task the safety controller must keep the Inlets locked until the safety operator decides to select the reset option. These simple tasks can be specified by modes, and this would result in the mode graph below (figure C.7).

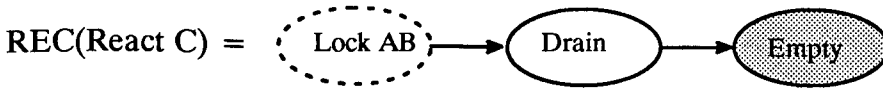


Figure C.7. Recovery graph

ii. In this step, we extend PF to cover the mode graph REC(React C).

An analysis similar to that for the start up graph gives the following preconditions:

$$PF(\text{Lock AB}) = p_8 < E_{act} - U4 \cdot \Delta T_m \wedge p_6 + 2 \cdot p_5 < T_{vol} - 2 \cdot U4 \cdot F_{maxD}.$$

$$PF(\text{Drain}) = (p_8 \leq E_{act} - (RT - U4 \cdot \Delta T_m) \wedge p_6 + 2 \cdot p_5 < T_{vol}).$$

$$PF(\text{Empty}) = p_7 = 0.$$

iii. In this step we construct the mode graph REC(React C).

Lock AB mode

The safety controller is in this mode while LockA and LockB are being turned on, and SValve is being opened. At the start of the mode the Thermometer reading will be at least

Eact-($\Delta T_p + RT$). During this mode *SafeLight* is at amber or red and *LockD* is on. The safety controller must set *SafeLight* to red, close the locks, open *SV*Valve and then must leave the mode. The safety controller must spend no more than *U4* seconds in this mode.

Lock AB =

$\langle p_{30} \geq \text{Eact}-(\Delta T_p + RT), p_{15} \in \{a, r\} \wedge p_{28} = \text{on}, p_{15} = r \wedge \text{LRset} \wedge p_{29} = \text{Smax}, 0, U4 \rangle$.

Drain mode

The safety controller is in this mode while the vessel is being drained. At the start of the mode, *SafeLight* will be red. During this mode, *SafeLight* is at green or red, the locks must remain on, and *SV*Valve open. The safety controller must set *Safelight* to green and then leave the mode. The safety controller must spend between *L1* and *U5* seconds in the mode.

Drain = $\langle p_{15} = r, p_{15} \in \{g, r\} \wedge \text{LRset} \wedge p_{29} = \text{Smax}, p_{15} = g, L1, U5 \rangle$.

Empty mode

The safety controller is in this mode while the vessel is empty. During this mode, *Safelight* is at green and the locks must remain on. The safety controller must leave this mode as soon as *Safety select* is at reset or off.

Empty = $\langle \text{true}, p_{15} = g \wedge \text{LRset}, p_{13} \in \{\text{reset}, \text{off}\} \rangle$.

e. In this step the completeness and consistency of the recovery graph must be confirmed. In this example, we skip this step.

f. The proofs for production rule *a* for the (*Lock AB*, *Drain*) and (*Drain*, *Empty*) arcs are sketched below.

Lock AB, Drain arc.

For rule *a* we show:

$H \text{ sat } (p_8 \leq \text{Eact} - U4 \cdot \Delta T_m \wedge p_6 + 2 \cdot p_5 < T_{vol} - 2 \cdot U4 \cdot F_{maxD}) @ s(\text{Int}) \wedge H \text{ sat } \text{Lock AB} @ \text{Int}$
 $\Rightarrow H \text{ sat } (p_8 \leq \text{Eact} - (RT - U4 \cdot \Delta T_m) \wedge p_6 + 2 \cdot p_5 < T_{vol}) @ e(\text{Int})$.

Proof. $p_6 + 2 \cdot p_5 < T_{vol} - \Delta V_p @ s(\text{Int})$ and since *LockD* is on during *Lock AB*, from relations *Ir*₈ and *Hr*₇, $H \text{ sat } p_6 + 2 \cdot p_5 < T_{vol} - \Delta V_p @ s(\text{Int})$. From the upper bound of this mode and *Hr*₅ the temperature at *e*(*Int*) can be at most $U4 \cdot \Delta T_m$ greater than the temperature at *s*(*Int*), if $U4 < RT / \Delta T_m$.

Drain, Empty arc.

For rule *a* we show: $H \text{ sat } \text{Drain}@Int \Rightarrow H \text{ sat } p_7=0@e(Int)$.

Proof. From the lower bound of *Drain* $\text{dur}(Int)$ must be at least $L1$; from the invariant of *Drain* the Inlet valves are locked and *SValve* is open during *Int*. Hence, from Ir_2, Ir_6, Ir_7, Ir_8 and Hr_1 , $H \text{ sat } p_7=0@e(Int)$, if the following history predicate is a history relation: $\text{dur} \geq L1 \wedge \forall t: p_{19}(t)=0 \wedge p_{29}(t)=S_{\max} \Rightarrow p_{7,1}=0$.

The proofs for production rule *b* for the *Lock*, *Drain* and *Empty* modes are given below.

Lock AB mode.

For rule *b* we show:

$H \text{ sat } (p_8 \leq E_{act} \wedge p_6 + 2.p_5 < T_{vol} - \Delta V_p) @s(Int) \wedge H \text{ sat } \text{Lock AB}@Int$
 $\Rightarrow H \text{ sat } (p_8 < E_{act} \wedge p_6 < T_{vol})@Int$.

Proof. This result follows from the arguments given for production rule *a* of *Lock AB*.

Drain mode.

For rule *b* we show:

$H \text{ sat } (p_8 \leq E_{act} - (RT - U_4 \cdot \Delta T_m) \wedge p_6 + 2.p_5 < T_{vol}) \wedge H \text{ sat } \text{Drain}@Int$
 $\Rightarrow H \text{ sat } (p_8 < E_{act} \wedge p_6 < T_{vol}) \vee (p_8 < T_{act})@Int$.

Proof. Since *LockD* is on during *Int*, if $p_6 + 2.p_5 < T_{vol}@s(Int)$ from Ir_8 and Hr_7 , $p_6 + 2.p_5 < T_{vol}$ during *Int*, hence $H \text{ sat } p_6 < T_{vol}@Int$. From the upper bound of *Drain* and the relation Hr_5 the maximum rise in the temperature from $s(Int)$ to $e(Int)$ is bounded by $U_4 \cdot \Delta T_m$. Hence, $H \text{ sat } p_8 < E_{act} - (RT - U_4 \cdot \Delta T_m)@s(Int) \Rightarrow H \text{ sat } p_8 < E_{act}@e(Int)$, if $RT > (U_4 + U_5) \cdot \Delta T_m$.

Empty mode.

For rule *b* we show:

$H \text{ sat } p_7=0@s(Int) \wedge H \text{ sat } \text{Empty}@Int \Rightarrow H \text{ sat } p_7=0@Int$.

Proof. This result follows from the invariant from the fact that the locks are on during *Empty*.

g. Production rule *c* holds for (*React C*, *Lock AB*) since $PF(\text{Lock AB})$ holds during *React C*.

h. In this step, we step we check that the modes of set REM are complete. However, since we only constructed one recovery graph we will skip this step.

Step 4 (Reset phase)

a. Let us suppose that for this example no reset phase is required. Hence we skip Step 4.

Step 5 (Shut Down Phase and End Phase)

a. Let us suppose that an analysis of the end phase of the chemical plant leads to the definition of the following mode, precondition and shut down condition.

End Control Mode

The safety controller is in this mode when it will no longer enter the monitor phase. At the start of this mode Safety Select will be at closed. During this mode, Safelight is at green and the locks are on. The safety controller leaves this mode as soon as the termination predicate is satisfied.

End Control = $\langle p_{13} = \text{off}, p_{15} = g \wedge \text{LRset}, \Omega \rangle$.

PF(End Control) = $p_7 = 0$.

The shut down condition is $p_{13} = \text{off}$.

b. In this step, the shut down modes must be identified. For simplicity we assume that there is only one shut down mode: *Empty*.

c. For the chemical plant no shutdown graph is required – we can simply add an arc from *Empty* to End control.

Since no shut down graph is necessary we can skip the steps d, e and f.

g. Production rule *c* holds for the pair (Empty, End Control) since the vessel is empty at the end of the *Empty* mode.

Mode Graph Connection

The SCS of the chemical plant is produced by connecting the mode graphs of the phases of the safety controller, by applying the function CSCS (see algorithm 8.1). The resultant graph is given in figure C.8.

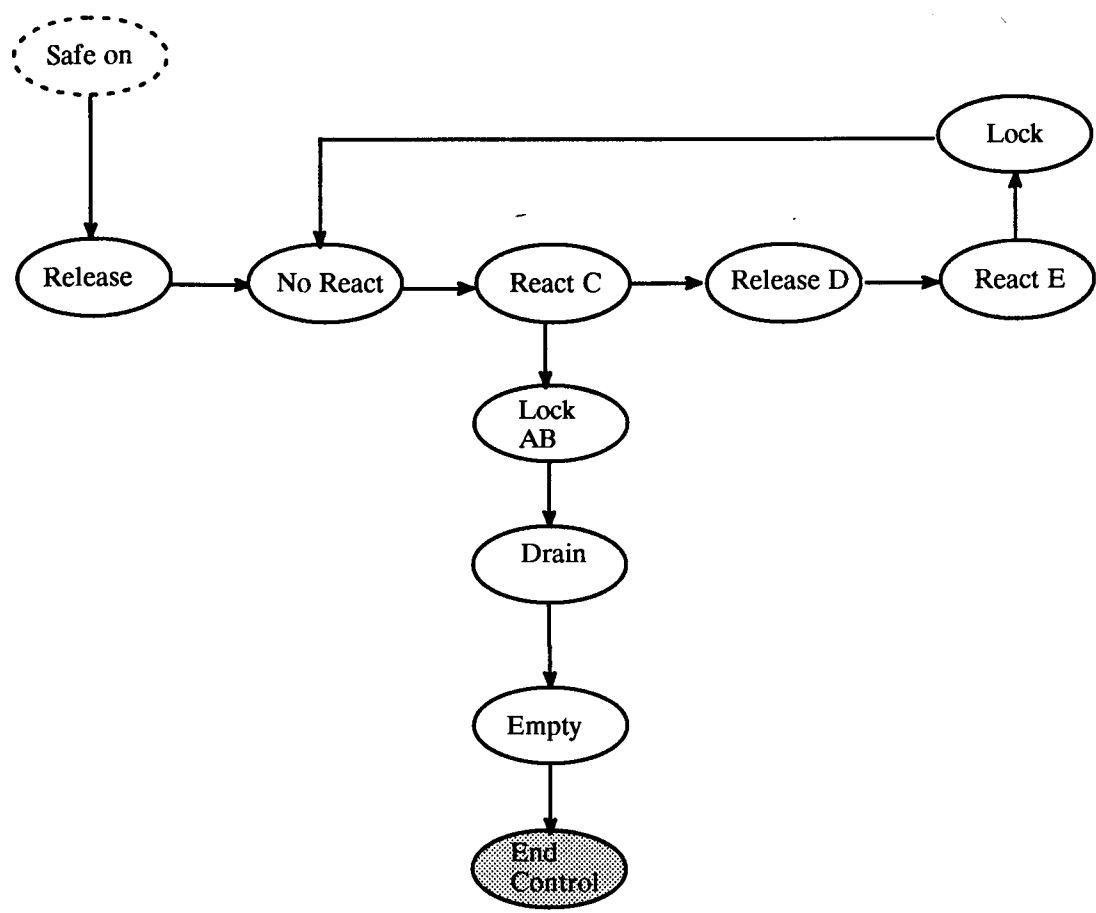


Figure C.8. Safety Controller Specification

C.3. Mission Controller Analysis

In this section, I will provide an overview of the mission controller analysis of the chemical plant. This analysis will lead to the production of MED(CP) and MCS(CP).

C.3.1 Mission Environment Description

Here the production of the mission environment description, by following the guidelines of section 8.4, is discussed.

Step 1

Let us suppose that the actuators of the mission controller consist of a valve for each inlet, a temperature regulator with power and temperature settings and a valve for OutletE. The variables that model the state of these actuators are given in table C.5.

Table C.5: Mission Controller Variables

Notation	Units	Name	Comments
p32	mm	ValveA	The extent to which the valve for InletA is open.
p33	mm	ValveB	As p32, for InletB.
p34	mm	ValveD	As p32, for InletD.
p35	Reg_Set	RegSwitch	The position of the switch for the regulator.
p36	Reg_Temp	RegTemp	The setting of the regulator control temperature.
p37	mm	EValve	As p32, for OutletE.

Step 2

An analysis to identify the ranges and classes of the actuators of the mission controller is performed; the results of this analysis are given in tables C.6 and C.7.

Table C.6: Ranges of Mission Controller Variables

Notation	Range	Comments
p32	$\{x \in \mathbb{R} \mid 0 \leq x \leq A_{\max}\}$	Where A_{\max} is the maximum width to which the valve for Inlet A can be opened.
p33	$\{x \in \mathbb{R} \mid 0 \leq x \leq B_{\max}\}$	As p32, for InletB.
p34	$\{x \in \mathbb{R} \mid 0 \leq x \leq D_{\max}\}$	As p32, for InletD.
p35	{off, on}	Where off and on are the two states of the regulator switch
p36	{CL, Y, Z}	Where CL, Y, Z are the three settings of the regulator.
p37	$\{x \in \mathbb{R} \mid 0 \leq x \leq E_{\max}\}$	As p32, for OutletE.

Table C.7: Categories and Classes of Mission Controller Variables

Variables	Category	Class	Comments
p32, p33, p34, p37	Controller	Continuous	The valves are continuous variables.
p35, p36	Controller	Free	No restrictions are imposed on the regulator.

Step 3

An analysis to identify the basic relationships is performed; the results of this analysis are given in table C.8.

Table C.8: Relationships of Mission Controller Variables

No.	Related variables	Relationship	Comments
Ir13	p16, p32	$p16 = F_I(p32)$	The flow rate of A into the vessel is given by applying the function F_I to the value of ValveA.

Ir ₁₄	p ₁₇ , p ₃₃	$p_{17} = F_1(p_{33})$	As Ir ₁₃ , for ValveB.
Ir ₁₅	p ₁₈ , p ₃₄	$p_{18} = F_1(p_{34})$	As Ir ₁₃ , for ValveD.
Ir ₁₆	p ₁ , p ₁₂ , p ₃₂ , p ₃₃ , p ₃₄ , p ₃₇	$p_1 = s(T) \Rightarrow$ $p_{12} = \text{off} \wedge p_{32} = 0 \wedge p_{33} = 0 \wedge p_{34} = 0$ $\wedge p_{37} = 0.$	At the start of the system lifetime the vessel Plant select is off and the valves closed.
Hr ₁₅	p ₈ , p ₃₅ , p ₃₆	$\forall t: (p_{36}(t) = \text{on} \wedge p_{35}(t) = p_{35,0})$ $\wedge \text{dur} > RS$ \Rightarrow $\forall t: (RS): p_{35}(t) - p_8(t) \leq \Delta R)$	If the regulator is on for an interval during which regulator set is constant and the duration of the interval is greater than RS then after the first RS seconds of that interval the temperature is approximately the set value.

Step 4

By inspecting the relations of the mission controller (table C.8) and those of the safety controller (table C.4) we infer that the relationships involving the flow of liquid into the vessel are influenced by the state of the locks of the safety controller. The dependent relations are given in table C.9. Hence the relations Ir₁₃, Ir₁₄ and Ir₁₅ (of table C.8) are replaced by the corresponding relations of table C.9.

Table C.9: Dependent Relationships of Mission Controller Variables

No.	Related variables	Relationship	Comments
Ir ₁₃	p ₁₆ , p ₂₆ , p ₃₂	$p_{26} = \text{off} \Rightarrow p_{16} = F_1(p_{32})$	The flow rate of A into the vessel is given by applying the function F_1 to the value of ValveA, provide LockA is off.
Ir ₁₄	p ₁₇ , p ₂₇ , p ₃₃	$p_{27} = \text{off} \Rightarrow p_{17} = F_1(p_{33})$	As Ir ₁₃ , for ValveB.
Ir ₁₅	p ₁₈ , p ₂₈ , p ₃₄	$p_{28} = \text{off} \Rightarrow p_{18} = F_1(p_{34})$	As Ir ₁₄ , for ValveD.

The mission environment description of the chemical plant is given below:

$$\text{MED}(\text{CP}) = \langle T, \langle p_1, \dots, p_{37} \rangle, \langle Cp_1, \dots, Cp_{38} \rangle, \langle Ir_1, \dots, Ir_8, Ir_{13}, \dots, Ir_{16} \rangle, \langle Hr_1, \dots, Hr_{14} \rangle \rangle.$$

C.3.2. Monitor Relations

In this section the guidelines to identify the monitor relations are followed.

Step 1 (Monitor Phase Relations)

a. The safety controller variables that influence the mission controller are the three locks.

Hence $DV = p_{26}, p_{27}$ and p_{28} .

b. The behaviour of the safety controller during the monitor phase is investigated to identify any relations over the locks. The relations are given in table C.10.

Table C.10: Monitor Phase Relations over Locks

No.	Related variables	Relationship -	Comments
Ir ₁₇	p ₂₆ , p ₂₇ , p ₂₉	$p_{26} = \text{off} \wedge p_{27} = \text{off} \wedge p_{29} = 0$	LockA and LockB are off and SValve is closed during the monitor phase.
Hr ₁₄	p ₇ , p ₂₈	$p_{28,0} = \text{off} \wedge \forall t: p_7(t) > 2.\Delta Vp \Rightarrow \forall t: p_{28}(t) = \text{off}$	During the monitor phase if LockD is off at the start of an interval and for any time point in that interval the volume of liquid in the vessel is greater than 2.ΔVp then for any time point in that interval LockD is off. This relation follows from an analysis of the React E mode.

Step 2 (Monitor Controller Variables)

a. For the mission controller to perform its mission, it must be able to load chemicals A, B and D into the vessel. The relation Ir₁₆ allows the mission controller to have full control of the flow of A and B into the vessel. However, the mission controller will have full control of the flow of D into the vessel only during the intervals that satisfy the antecedent of relation Hr₁₄ (since when lockD is closed no D can flow into the vessel). Hence to “detect” such intervals a sensor which allows the mission controller to monitor the status of LockD is required.

b. Let us suppose that analysis of the system shows that it is feasible to have a sensor which detects the position of LockD without interfering with the behaviour of the safety controller.

We introduce a variable LkSensor (p₃₈) which allows the mission controller to monitor the status of LockD. This variable has the range {off, mid, on} and the class free. We capture the fact that LkSensor monitors LockD with the following invariant relation (Ir₁₇):

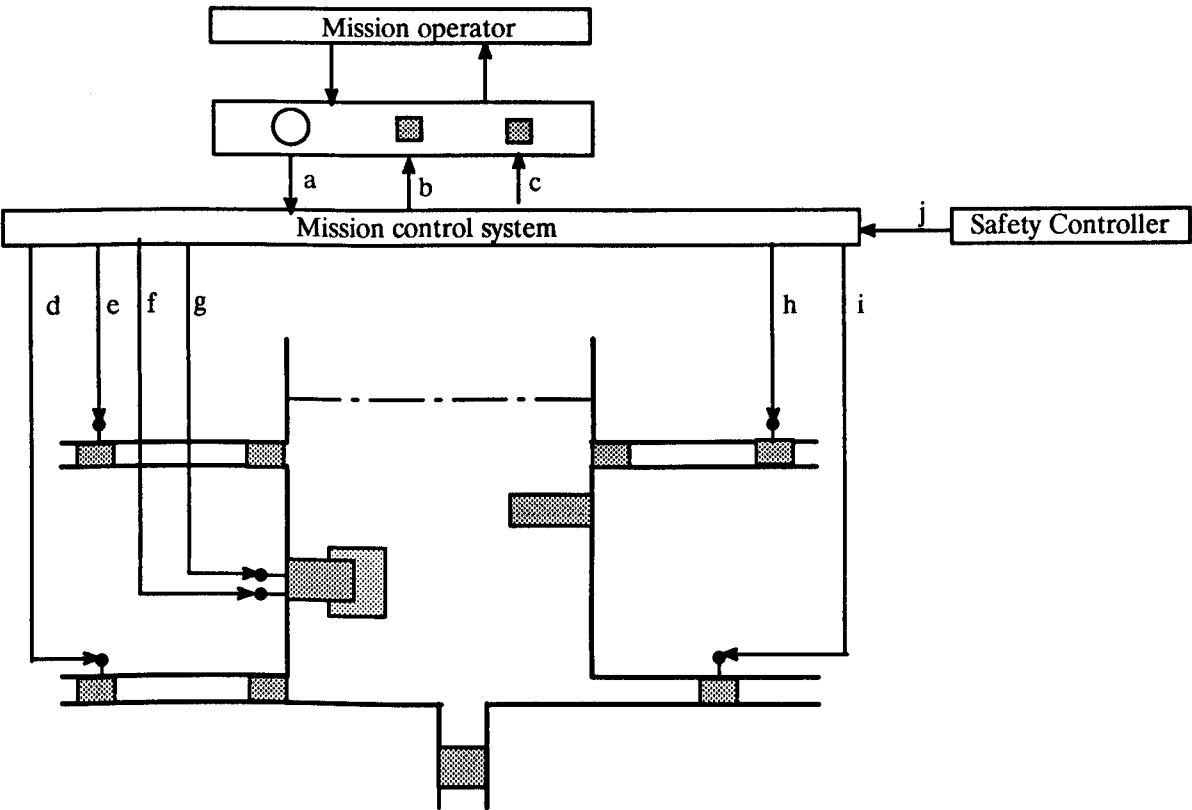
$p_{38} = p_{28}$.

Step 3 (Modify Environment Description)

The modified mission environment description of the chemical plant is given below:

$MED(CP) = \langle T, \langle p_1, \dots, p_{38} \rangle, \langle Cp_1, \dots, Cp_{38} \rangle, \langle Ir_1, \dots, Ir_8, Ir_{13}, \dots, Ir_{17} \rangle, \langle Hr_1, \dots, Hr_{14} \rangle \rangle$.

The interactions between the components of the mission controller, the mission operator, safety controller and plant are illustrated in figure C.9.



Key

- a: Plant select
- b: Indicator
- c: ELight
- d: ValveB
- e: ValveA
- f: RegSet
- g: RegTemp
- h: ValveD
- i: ValveE
- j: LkSensor

Figure C.9. Chemical Plant Mission Controller

C.3.3. Mission Controller Specification Analysis.

Here the production of the MCS, by following the guidelines of section 8.5.2, is discussed. The structure of MRS(CP) is given in figure C.10. The specifications of the modes are given in appendix B.

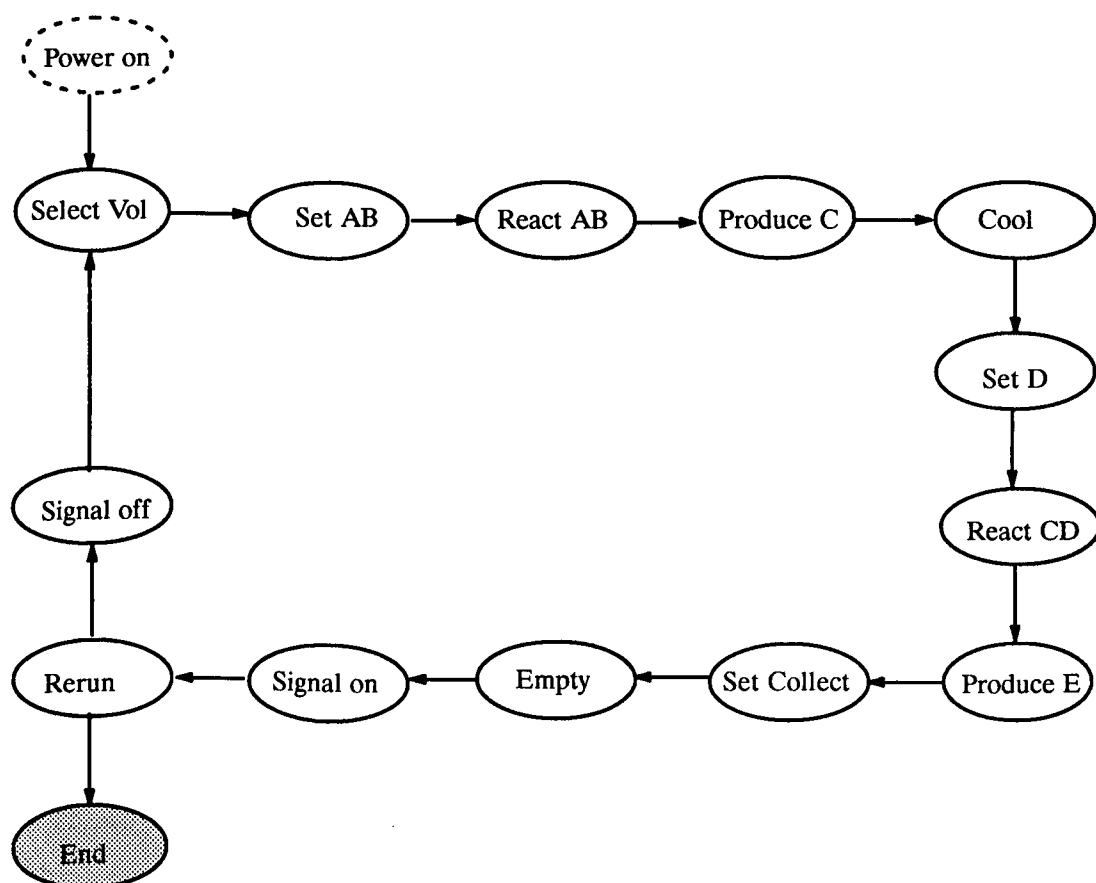


Figure C.10. Mission Real World Specification of Chemical Plant

Construct Outline Specification

In this preliminary analysis stage, a precondition function, template function and delay function are constructed for MRS(CP).

Step 1 (Precondition Function)

The precondition of a mode is defined by an analysis of the start and invariant predicates of that mode, to identify the conditions which must hold at the start of the mode. A typical

example, the *Power on* mode, is discussed next.

$\text{Power on} = \langle p_{12} = \text{off}, p_7 = 0 \wedge p_{12} \in \{\text{off}, \text{on}\} \wedge p_{14} = g \wedge p_{25} = \text{off}, p_{12} = \text{on} \rangle.$

From the start predicate $p_{12} = \text{off}$ is satisfied by the state at the start of this mode and from the invariant predicate $p_7 = 0 \wedge p_{14} = g \wedge p_{25} = \text{off}$ is satisfied by the state at the start of the mode. Hence, $\text{PF}(\text{Power on})$ is defined as $p_7 = 0 \wedge p_{12} = \text{off} \wedge p_{14} = g \wedge p_{25} = \text{off}$.

A similar analysis can be performed to derive the preconditions of the other modes. As another example, the precondition of *Set AB* is presented.

$\text{PF}(\text{Set AB}) = p_7 = 0 \wedge p_8 < Y \wedge p_{12} = \text{start} \wedge p_{14} = g \wedge p_{18} = 0 \wedge p_{25} = \text{off}.$

Step 2 (Template Function)

The template predicate of a mode is defined by an analysis of the precondition of that mode and the relations of the mission controller variables. A typical example, the *Set AB* mode, is discussed next.

$\text{PF}(\text{Set AB}) = p_7 = 0 \wedge p_8 < Y \wedge p_{12} = \text{start} \wedge p_{14} = g \wedge p_{18} = 0 \wedge p_{25} = \text{off}.$

Firstly, we consider the state of the valves. Since $p_7 = 0$ the inlet valves must be closed, but no constraint is imposed on *EValve*. Secondly, we consider the regulator since the vessel contains no liquid, and the temperature is less than *Y*, the regulator should be off. Hence

$\text{TF}(\text{Set AB}) = \text{IVC} \wedge p_{35} = \text{off},$ where $\text{IVC} = p_{32} = 0 \wedge p_{33} = 0 \wedge p_{34} = 0;$

Step 3 (Delay Function)

The delay of a mode is defined by an analysis of the end predicate of that mode. At this early stage the analysis can only determine if the delay will be zero (or non-zero). A typical example of a mode with a zero delay, *Power on* is discussed next.

$\text{Power on} = \langle p_{12} = \text{of}, p_7 = 0 \wedge p_{12} \in \{\text{of}, \text{on}\} \wedge p_{14} = g \wedge p_{25} = \text{off}, p_{12} = \text{on} \rangle.$

The end predicate of *Power on* is $p_{12} = \text{on}$; since the mission controller specification can define an end predicate over p_{12} , a controller graph with a zero delay can be defined for this mode (see *Control on*, below).

Construct Controller Function

In this section, the controller function is defined over the modes of MRS(CP). With the mode specifications, conditions over the time bounds of the controller graph (which arise from the time bounds of the modes of MRS(CP)) are also presented.

Power on

$$CF(\text{Power on}) = \text{Control on}$$

Control on Mode

The mission controller starts in this mode. At the start of this mode, Plant select will be at off. During this mode, Plant select is at off or on, RLight is at green, ELight is off, the inlet valves are closed and Regulator is off. The mission controller must leave this mode as soon as Plant select is at on.

Power on = $\langle p_{12} = \text{off}, p_{12} \in \{\text{off}, \text{on}\} \wedge p_{14} = g \wedge p_{25} = \text{off} \wedge \text{IVC} \wedge p_{35} = \text{off}, p_{12} = \text{on} \rangle$,
 where $\text{IVC} = (p_{32} = 0 \wedge p_{33} = 0 \wedge p_{34} = 0)$.

Select Vol

$$CF(\text{Select Vol}) = \text{Required Vol}$$

Required Vol Mode

The mission controller is in this mode while the operator selects the volumes of A, B and D required for the production of C and E. At the start of this mode Plant select is at on. During this mode, Plant select is at on or start, RLight is green, ELight is off, the inlet valves are closed and the Regulator is off. The mission controller must leave this mode as soon as Plant select is at start.

Required Vol = $\langle p_{12} = \text{on}, p_{12} \in \{\text{on}, \text{start}\} \wedge p_{14} = g \wedge p_{25} = \text{off} \wedge \text{IVC} \wedge p_{35} = \text{off}, p_{12} = \text{start} \rangle$.

Set AB



During this mode graph, Plant select is at start, ELight is at off, ValveD is closed and the regulator off. $\text{Inv}(\text{CF}(\text{Set AB})) = p_{12} = \text{start} \wedge p_{25} = \text{off} \wedge p_{34} = 0 \wedge p_{35} = \text{off}$. This will be abbreviated to InvRAB.

Valve AB Mode

The mission controller is in this mode while ValveA and ValveB are opened to fill the vessel with A and B, and EValve is closed. At the start of this mode ValveA, ValveB and ValveD are closed. During this mode, InvAB holds and Indicator is at green. The mission controller must open ValveA to the extent required by function f_A and ValveB to the extent required by the function f_B , close EValve and then leave the mode. The mission controller must spend at most U1 seconds in the mode.

Valve AB = $\langle \text{IVC}, \text{InvAB} \wedge p_{14} = g, \text{SV}, 0, \text{U1} \rangle$,

where $\text{SV} = p_{32} = f_A(p_9, p_{10}) \wedge p_{33} = f_B(p_9, p_{10}) \wedge p_{37} = 0$.

Load AB Mode

The mission controller is in this mode while the required volumes of A and B are being loaded into the vessel. During this mode, InvAB holds, the Indicator is at green and EValve is closed. The mission controller must start to close the valves and then leave this mode. The mission controller must spend between $\text{Lt}(p_9, p_{10})$ and $\text{Lt}(p_9, p_{10}) + \Delta L$ seconds in the mode.

Load AB = $\langle \text{true}, \text{InvAB} \wedge p_{14} = g \wedge p_{37} = 0, \text{CAB}, \text{Lt}(p_9, p_{10}), \text{Lt}(p_9, p_{10}) + \Delta L \rangle$,

where $\text{CAB} = p_{32} \neq f_A(p_9, p_{10}) \vee p_{33} \neq f_B(p_9, p_{10})$.

Reset AB Mode

The mission controller is in this mode while the valves are being reset. At the start of this mode the Indicator is at green. During this mode, InvAB holds, the Indicator is at green or amber, EValve is closed, and ValveA and ValveB are not both closed or the indicator is at amber. The mission controller must close ValveA and ValveB, set the Indicator to amber and then leave this mode. The mission controller must spend at most U2 seconds in the mode.

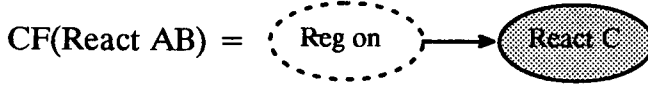
Reset AB =

$\langle p_{14} = g, \text{InvAB} \wedge p_{14} \in \{g, a\} \wedge p_{37} = 0 \wedge (p_{32} \neq 0 \vee p_{33} \neq 0 \vee p_{14} = a), \text{VC} \wedge p_{14} = a, 0, \text{U2} \rangle$,

where $\text{VC} = (\text{IVC} \wedge p_{37} = 0)$.

Timing Constraint: $U1 + U2 + Lt(p_9, p_{10}) + \Delta L < Su(p_9, p_{10})$.

React AB



During this mode graph, Plant select is at start, ELight is at off and ValveA, ValveB, ValveD and EValve are closed. $Inv(CF(React AB)) = p_{12} = start \wedge p_{25} = off \wedge VC$. This will be abbreviated to InvRAB.

Reg on Mode

The mission controller is in this mode while the regulator is being set. At the start of this mode the Regulator is off. During this mode, InvRAB holds and RLight is amber. The mission controller must set the Regulator to Y and then leave this mode. The mission controller must spend at most U3 seconds in this mode.

$Reg\ on = \langle p_{35} = off, InvRAB \wedge p_{14} = a, RY, 0, U3 \rangle$, where $RY = (p_{35} = on \wedge p_{36} = Y)$.

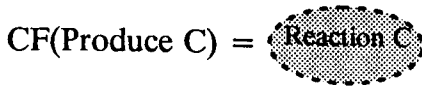
React C Mode

The mission controller is in this mode while a reaction to produce C is being activated. During this mode, InvRAB holds, RLight is amber or red, and the Regulator is set at Y. The mission controller must turn RLight to red and then leave this mode. The mission controller must spend between RS and $RS + \Delta RS$ seconds in the mode. (Hr_{13} ensures the temperature is at Y.)

$React\ C = \langle true, InvRAB \wedge p_{18} \in \{a, r\} \wedge RY, p_{14} = r, RS, RS + \Delta RS \rangle$.

Timing Constraint: $U3 + RS + \Delta RS < SH$.

Produce C



Reaction C Mode

The mission controller is in this mode while a reaction to produce C is in progress. During this mode, InvRAB holds, RLight is amber or red and the regulator is set at Y. The mission controller must set RLight to green and then leave this mode. The mission controller must

spend between $CL(p_9, p_{10})$ and $CL(p_9, p_{10}) + \Delta C$ seconds in this mode.

Reaction C = $\langle \text{true}, \text{InvRAB} \wedge p_{14} \in \{a, r\} \wedge \text{RY}, p_{14}=a, CL(p_9, p_{10}), CL(p_9, p_{10}) + \Delta C \rangle$.

where $\text{RC} = (p_{35}=\text{on} \wedge p_{36} = \text{CL})$.

Cool



RegC on Mode

The mission controller is in this mode while the regulator is being set to C. During this mode, *InvRAB* holds, *RLight* is amber and the Regulator is set at CL or Y. The mission controller must set the regulator to CL and then leave this mode. The mission controller must spend at most U4 seconds in the mode.

$\text{RegC on} = \langle \text{true}, \text{InvRAB} \wedge p_{14}=a \wedge \text{RC} \vee \text{RY}, \text{RC}, 0, \text{U4} \rangle$,

where $\text{RC} = (p_{35}=\text{on} \wedge p_{36} = \text{CL})$.

Cool down Mode

The mission controller is in this mode while the vessel is being cooled down after a reaction. During this mode, *InvRAB* holds, *RLight* is green or red and the Regulator is set at CL. The mission controller must turn *RLight* to green; and then leave this mode. The mission controller must spend between RS and $RS + \Delta RS$ seconds in the mode.

$\text{Cool down} = \langle \text{true}, \text{InvRAB} \wedge p_{14} \in \{g, r\} \wedge \text{RC}, p_{14}=g, \text{RS}, \text{RS} + \Delta \text{RS} \rangle$.

Timing Constraint: $\text{U4} + \text{RS} + \Delta \text{RS} < \text{Cu}$.

Set D



During this mode graph, Plant select is at start, *ELight* is off, *ValveA* and *ValveB* are Closed, Regulator is set at CL and *EValve* is Closed.

$\text{Inv}(\text{CF}(\text{Set D})) = p_{12}=\text{start} \wedge p_{25}=\text{off} \wedge p_{32}=0 \wedge p_{33}=0 \wedge \text{RC} \wedge p_{37}=0$. This will be denoted by *InvD*.

Wait D Mode

The mission controller is in this mode until LkSensor is at off. During this mode, InvD holds and RLight is green. The mission controller must leave this mode as soon as LkSensor is at off.

The mission controller must spend at most U5 seconds in this mode.

Wait D = $\langle \text{true}, \text{InvD} \wedge p_{14} = g, p_{38} = 0, 0, U5 \rangle$.

Valve D Mode

The mission controller is in this mode while ValveD is opened to fill the vessel with the required volume of D. At the start of this mode LkSensor is at off. During this mode, InvD holds and RLight is green. The mission controller must open ValveD and then leave the mode. The mission controller must spend at most U6 seconds in this mode.

Valve D = $\langle p_{38} = \text{off}, \text{InvD} \wedge p_{14} = g, p_{34} = D_{\text{max}}, 0, U6 \rangle$.

Load D Mode

The mission controller is in this mode while the required volume of D is being loaded into the vessel. At the start of this mode ValveD is open. During this mode, InvD holds and RLight is green. The mission controller must start to close ValveD and then leave this mode. The mission controller must spend between $Ld(p_{11})$ and $Ld(p_{11}) + \Delta L$ seconds in this mode, where $Ld(p_{11})$ gives the time required to load $p_{11} \text{ dm}^3$ of D into the vessel (i.e. $F_1(D_{\text{max}}).Ld(p_{11}) = p_{11}$).

Load D = $\langle p_{34} = D_m, \text{InvD} \wedge p_{14} = g, p_{34} < D_m, Ld(p_{11}), Ld(p_{11}) + \Delta L \rangle$.

Reset D Mode

The mission controller is in this mode while the valves are being reset. During this mode, InvD holds and RLight is green or amber, ValveD is not closed or RLight is amber. The mission controller must close ValveD, then switch RLight to amber and then leave this mode. The mission controller must spend at most U7 seconds in the mode.

Reset D = $\langle \text{true}, \text{InvD} \wedge p_{14} \in \{g, a\} \wedge (p_{33} \neq 0 \vee p_{14} = a), VC \wedge p_{14} = a, 0, U7 \rangle$.

Timing Constraint: $U5 + U6 + U7 + Ld(p_{11}) + \Delta L + \Delta RS < SD$.

React CD

During this mode graph, Plant select is at start, ELight is off, ValveA, ValveB and EValve are closed.

$\text{Inv(React CD)} = p_{12} = \text{st} \wedge p_{25} = \text{off} \wedge \text{VC} \wedge p_{35} = \text{on}$. This will be abbreviated to InvCD .

RegE on Mode

The mission controller is in this mode while the regulator is being set to Z. At the start of this mode the Regulator is set at CL. During this mode, InvCD holds, RLight is amber, and the Regulator is set at CL or Z. The mission controller must set the Regulator to Z and then leave this mode. The mission controller must spend at most U8 seconds in this mode.

$\text{RegE on} = \langle \text{RC}, \text{InvCD} \wedge p_{14} = a \wedge (\text{RZ} \vee \text{RY}), \text{RZ}, 0, \text{U8} \rangle$,

where $\text{RZ} = (p_{35} = \text{on} \wedge p_{36} = \text{Z})$.

React E Mode

The mission controller is in this mode while a reaction to produce D is being activated. During this mode, InvCD holds, RLight is amber or red and the Regulator is set at Z. The mission controller must set RLight to red and then leave this mode. The mission controller must spend between RS and $\text{RS} + \Delta\text{RS}$ seconds in the mode.

$\text{React E} = \langle \text{true}, \text{InvCD} \wedge p_{14} \in \{a, r\} \wedge \text{RZ}, p_{14} = r, \text{RS}, \text{RS} + \Delta\text{RS} \rangle$.

Timing Constraint: $\text{U8} + \text{RS} + \Delta\text{RS} < \text{Ru}$.

Produce E

$$\text{CF(Produce E)} = \text{Reaction E}$$

Reaction E mode

The mission controller is in this mode while a reaction to produce E is in progress. During this mode InvCD holds, RLight is green or red and the Regulator is set at Z. The mission controller must set RLight to green and then leave this mode. The mission controller must spend between $\text{DL}(p_{11})$ and $\text{DL}(p_{11}) + \Delta\text{D}$ seconds in this mode.

$\text{Reaction E} = \langle p_{14} = r, \text{InvCD} \wedge p_{14} \in \{g, r\} \wedge \text{RZ}, p_{14} = g, \text{DL}(p_{11}), \text{DL}(p_{11}) + \Delta\text{Pd} \rangle$.

Set Collect

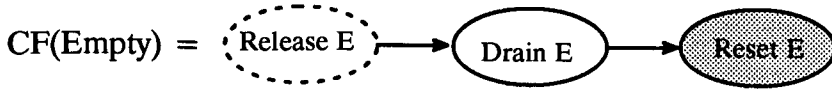
$$CF(\text{Set Collect}) = \text{Collect}$$

Collect mode

The mission controller is in this mode until the operator selects collect. At the start of this mode, Plant select is at start. During this mode Plant select is at Start or Collect, the Indicator is at green, ELight is off, ValveA, ValveB, ValveD and EValve are closed and the Regulator is set at Z. The mission controller must leave this mode as soon as Plant select is at collect.

$$\text{Collect} = \langle p_{12} = \text{start}, p_{12} \in \{\text{start}, \text{collect}\} \wedge p_{14} = g \wedge VC \wedge RZ, p_{12} = \text{collect} \rangle.$$

Empty



During this mode graph, Plant select is at collect, RLight is at green, ELight is off, ValveA, ValveB and ValveD are closed.

$\text{Inv}(\text{CF}(\text{Empty})) = p_{12} = \text{collect} \wedge p_{14} = g \wedge p_{25} = \text{off} \wedge \text{IVC}$. This will be abbreviated to InvEM .

Release E mode

The mission controller is in this mode while EValve is being opened. At the start of this mode, EValve is closed. During this mode InvEM holds and the Regulator is at Z. The mission controller must open EValve and then leave this mode. The mission controller must spend at most U9 seconds in the mode.

$$\text{Release E} = \langle p_{37} = 0, \text{InvEM} \wedge RZ, p_{37} = E_{\text{max}}, 0, U9 \rangle.$$

Drain E mode

The mission controller is in this mode while the vessel is emptied. At the start of this mode EValve is fully open. During this mode, InvEM holds and the Regulator is at Z. The mission controller must start to close EValve and then leave this mode. The mission controller must spend between EM and EM + ΔE seconds in the mode, where EM is the maximum time required to empty the vessel when EValve is open and Inlet valves are closed.

$$\text{Drain E} = \langle p_{37} = E_{\text{max}}, \text{InvEM}, p_{37} < E_{\text{max}} \text{ EM}, \text{EM} + \Delta E \rangle.$$

Reset E mode

The mission controller is in this mode while EValve is closed. During this mode, InvEM holds.

The mission controller must close EValve and turn the Regulator off and then leave this mode.

The mission controller must spend at most U10 seconds in this mode.

Reset E = $\langle \text{true}, \text{InvEM}, \text{VC} \wedge p_{35} = \text{off}, 0, \text{U10} \rangle$.

Timing Constraint: $\text{U9} + \text{U10} + \text{EM} + \Delta \text{E} < \text{Eu}$.

Signal on

$$\text{CF}(\text{Signal on}) = \text{ELight on}$$

ELight on mode

The mission controller is in this mode while ELight is being turned on. At the start of this mode ELight is off. During this mode, Plant select is at collect, RLight is at green, ValveA, ValveB, ValveD and EValve are closed and the Regulator is off. The mission controller must switch ELight on and then leave the mode. The mission controller must spend at most SGU seconds in this mode.

Signal on = $\langle p_{25} = \text{off}, p_{12} = \text{collect} \wedge p_{14} = \text{g} \wedge \text{VC} \wedge p_{35} = \text{off}, p_{25} = \text{on}, 0, \text{SGU} \rangle$.

Rerun

$$\text{CF}(\text{Rerun}) = \text{More E}$$

More E mode

The mission controller is in this mode while the operator decides if more E will be produced. At the start of this mode Plant select is at collect. During this mode, Plant select is at collect, off or on, RLight is green, ELight is on, ValveA, ValveB, ValveD and EValve are closed and the Regulator is off. The mission controller must leave this mode when Plant select is either off or on.

More E =

$\langle p_{12} = \text{co}, p_{12} \in \{\text{collect}, \text{off}, \text{on}\} \wedge p_{14} = \text{g} \wedge p_{25} = \text{on} \wedge \text{VC} \wedge p_{35} = \text{off}, p_{12} \in \{\text{off}, \text{on}\} \rangle$.

Signal off

$$CF(\text{Signal off}) = \text{ELight off}$$

ELight off

The mission controller is in this mode while ELight is being turned off. At the start of this mode ELight is on. During this mode, Plant select is at on, RLight is green, ValveA, ValveB, ValveD and EValve are closed and the Regulator is off. The mission controller must switch ELight off and then leave the mode. The mission controller must spend at most SGU seconds in this mode.

$$\text{ELight off} = \langle p_{25} = \text{on}, p_{12} = \text{on} \wedge p_{14} = g \wedge VC \wedge p_{35} = \text{off}, p_{25} = \text{off}, 0, \text{SGU} \rangle.$$

End

$$CF(\text{End}) = \text{Closed}$$

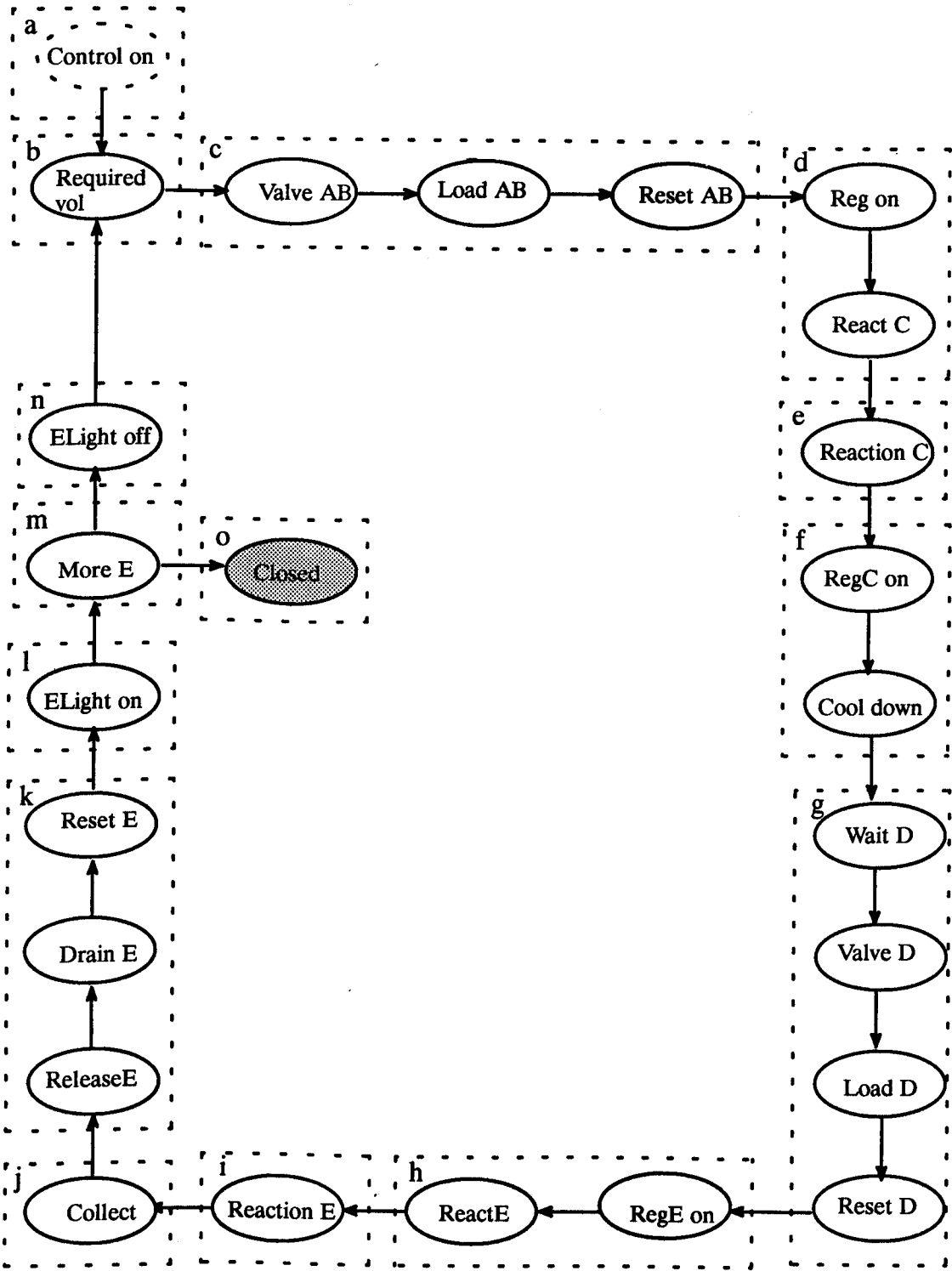
Closed Mode

The mission controller is in this mode when no more E will be produced. During this mode, Plant select is at off, RLight is green, ValveA, ValveB, ValveD and EValve are closed and the Regulator is off. The mission controller must leave this mode as soon as the termination predicate holds.

$$\text{Closed} = \langle \text{true}, p_{12} = \text{off} \wedge p_{14} = g \wedge VC \wedge p_{35} = \text{off}, \Omega \rangle.$$

Controller Graph Connection

The mission controller specification of the chemical plant is produced by applying the function MCGT to the function CF and mode graph MRS(CP). The resultant graph is given in figure C.11.



Key

- a: Power on b: Select Vol c: Set AB d: React AB e: Produce C f: Cool g: Set D
- h: React CD i: Produce E j: Set Collect k: Empty l: Signal on m: Rerun n: Signal off.
- o: End

Figure C.11. Mission Controller Specification

system must not be in a hazardous state. By the end of the phase, the collection of chemical E, must have been completed.

Phase Analysis

For the chemical plant, it is not necessary to decompose the high-level phases any further.

Mission Phase Specification Check

The mission phase graph is presented to the customer, who raises the following (new) point to the analyst: “The system should be able to perform a sequence of reactions”.

To allow multiple reactions during the system lifetime, the analyst modifies the phase graph, to that given in figure. B.3. The specification of the new phases is also refined by introducing the real world variables.

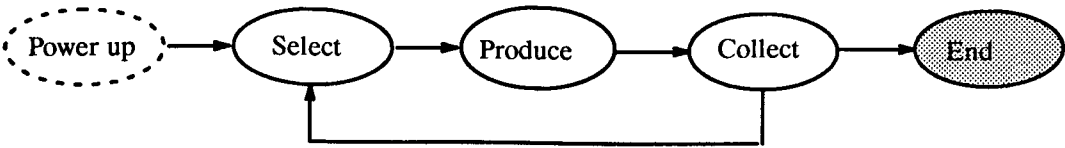


Figure B.3. Chemical Plant Mission Phase Specification

Power up phase

At the start of this phase the vessel will be empty. While the system is in the phase the vessel is empty and RLight is at green. The system leaves the phase as soon as Plant select is at on.

Select phase

At the start of this phase the vessel will be empty and Plant select at on. While the system is in this phase the vessel is empty and Plant select is at on or start and RLight is at green. The system leaves the phase as soon as the requested volumes of A, B and D have been selected.

Produce phase. As before.